PART

1

Foundations

In this part we introduce the fundamental components of an interactive system: the human user, the computer system itself and the nature of the interactive process. We then present a view of the history of interactive systems by looking at key interaction paradigms that have been significant.

Chapter 1 discusses the psychological and physiological attributes of the user, providing us with a basic overview of the capabilities and limitations that affect our ability to use computer systems. It is only when we have an understanding of the user at this level that we can understand what makes for successful designs. Chapter 2 considers the computer in a similar way. Input and output devices are described and explained and the effect that their individual characteristics have on the interaction highlighted. The computational power and memory of the computer is another important component in determining what can be achieved in the interaction, whilst due attention is also paid to paper output since this forms one of the major uses of computers and users' tasks today. Having approached interaction from both the human and the computer side, we then turn our attention to the dialog between them in Chapter 3, where we look at models of interaction. In Chapter 4 we take a historical perspective on the evolution of interactive systems and how they have increased the usability of computers in general.

1 THE HUMAN

OVERVIEW

- Humans are limited in their capacity to process information. This has important implications for design.
- Information is received and responses given via a number of input and output channels:
  – visual channel
  – auditory channel
  – haptic channel
  – movement.
- Information is stored in memory:
  – sensory memory
  – short-term (working) memory
  – long-term memory.
- Information is processed and applied:
  – reasoning
  – problem solving
  – skill acquisition
  – error.
- Emotion influences human capabilities.
- Users share common capabilities but are individuals with differences, which should not be ignored.

1.1 INTRODUCTION

This chapter is the first of four in which we introduce some of the 'foundations' of HCI. We start with the human, the central character in any discussion of interactive systems. The human, the *user*, is, after all, the one whom computer systems are designed to assist. The requirements of the user should therefore be our first priority.

In this chapter we will look at areas of human psychology coming under the general banner of *cognitive psychology*. This may seem a far cry from designing and building interactive computer systems, but it is not. In order to design something for someone, we need to understand their capabilities and limitations. We need to know if there are things that they will find difficult or, even, impossible. It will also help us to know what people find easy and how we can help them by encouraging these things. We will look at aspects of cognitive psychology which have a bearing on the use of computer systems: how humans perceive the world around them, how they store and process information and solve problems, and how they physically manipulate objects.

We have already said that we will restrict our study to those things that are relevant to HCI. One way to structure this discussion is to think of the user in a way that highlights these aspects. In other words, to think of a simplified *model* of what is actually going on. Many models have been proposed and it useful to consider one of the most influential in passing, to understand the context of the discussion that is to follow. In 1983, Card, Moran and Newell [56] described the *Model Human Processor*, which is a simplified view of the human processing involved in interacting with computer systems. The model comprises three subsystems: the perceptual system, handling sensory stimulus from the outside world, the motor system, which controls actions, and the cognitive system, which provides the processing needed to connect the two. Each of these subsystems has its own processor and memory, although obviously the complexity of these varies depending on the complexity of the tasks the subsystem has to perform. The model also includes a number of *principles of operation* which dictate the behavior of the systems under certain conditions.

We will use the analogy of the user as an information processing system, but in our model make the analogy closer to that of a conventional computer system. Information comes in, is stored and processed, and information is passed out. We will therefore discuss three components of this system: input–output, memory and processing. In the human, we are dealing with an intelligent information-processing system, and processing therefore includes problem solving, learning, and, consequently, making mistakes. This model is obviously a simplification of the real situation, since memory and processing are required at all levels, as we have seen in the Model Human Processor. However, it is convenient as a way of grasping how information is handled by the human system. The human, unlike the computer, is also influenced by external factors such as the social and organizational environment, and we need to be aware of these influences as well. We will ignore such factors for now and concentrate on the human's information processing capabilities only. We will return to social and organizational influences in Chapter 3 and, in more detail, in Chapter 13.

In this chapter, we will first look at the human's input–output channels, the senses and responders or effectors. This will involve some low-level processing. Secondly, we will consider human memory and how it works. We will then think about how humans perform complex problem solving, how they learn and acquire skills, and why they make mistakes. Finally, we will discuss how these things can help us in the design of computer systems.

1.2 INPUT–OUTPUT CHANNELS

A person's interaction with the outside world occurs through information being received and sent: input and output. In an interaction with a computer the user receives information that is output by the computer, and responds by providing input to the computer – the user's output becomes the computer's input and vice versa. Consequently the use of the terms input and output may lead to confusion so we shall blur the distinction somewhat and concentrate on the channels involved. This blurring is appropriate since, although a particular channel may have a primary role as input or output in the interaction, it is more than likely that it is also used in the other role. For example, sight may be used primarily in receiving information from the computer, but it can also be used to provide information to the computer, for example by fixating on a particular screen point when using an eyegaze system.

Input in the human occurs mainly through the senses and output through the motor control of the effectors. There are five major senses: sight, hearing, touch, taste and smell. Of these, the first three are the most important to HCI. Taste and smell do not currently play a significant role in HCI, and it is not clear whether they could be exploited at all in general computer systems, although they could have a role to play in more specialized systems (smells to give warning of malfunction, for example) or in augmented reality systems. However, vision, hearing and touch are central.

Similarly there are a number of effectors, including the limbs, fingers, eyes, head and vocal system. In the interaction with the computer, the fingers play the primary role, through typing or mouse control, with some use of voice, and eye, head and body position.

Imagine using a personal computer (PC) with a mouse and a keyboard. The application you are using has a graphical interface, with menus, icons and windows. In your interaction with this system you receive information primarily by sight, from what appears on the screen. However, you may also receive information by ear: for example, the computer may 'beep' at you if you make a mistake or to draw attention to something, or there may be a voice commentary in a multimedia presentation. Touch plays a part too in that you will feel the keys moving (also hearing the 'click') or the orientation of the mouse, which provides vital feedback about what you have done. You yourself send information to the computer using your hands, either by hitting keys or moving the mouse. Sight and hearing do not play a direct role in sending information in this example, although they may be used to receive information from a third source (for example, a book, or the words of another person) which is then transmitted to the computer.

In this section we will look at the main elements of such an interaction, first considering the role and limitations of the three primary senses and going on to consider motor control.

1.2.1 Vision

Human vision is a highly complex activity with a range of physical and perceptual limitations, yet it is the primary source of information for the average person. We can roughly divide visual perception into two stages: the physical reception of the stimulus from the outside world, and the processing and interpretation of that stimulus. On the one hand the physical properties of the eye and the visual system mean that there are certain things that cannot be seen by the human; on the other the interpretative capabilities of visual processing allow images to be constructed from incomplete information. We need to understand both stages as both influence what can and cannot be perceived visually by a human being, which in turn directly affects the way that we design computer systems. We will begin by looking at the eye as a physical receptor, and then go on to consider the processing involved in basic vision.

The human eye

Vision begins with light. The eye is a mechanism for receiving light and transforming it into electrical energy. Light is reflected from objects in the world and their image is focussed upside down on the back of the eye. The receptors in the eye transform it into electrical signals which are passed to the brain.

The eye has a number of important components (see Figure 1.1) which we will look at in more detail. The *cornea* and *lens* at the front of the eye focus the light into a sharp image on the back of the eye, the *retina*. The retina is light sensitive and contains two types of *photoreceptor*: *rods* and *cones*.

Rods are highly sensitive to light and therefore allow us to see under a low level of illumination. However, they are unable to resolve fine detail and are subject to light saturation. This is the reason for the temporary blindness we get when moving from a darkened room into sunlight: the rods have been active and are saturated by the sudden light. The cones do not operate either as they are suppressed by the rods. We are therefore temporarily unable to see at all. There are approximately 120 million rods per eye which are mainly situated towards the edges of the retina. Rods therefore dominate peripheral vision.

Cones are the second type of receptor in the eye. They are less sensitive to light than the rods and can therefore tolerate more light. There are three types of cone, each sensitive to a different wavelength of light. This allows color vision. The eye has approximately 6 million cones, mainly concentrated on the *fovea*, a small area of the retina on which images are fixated.

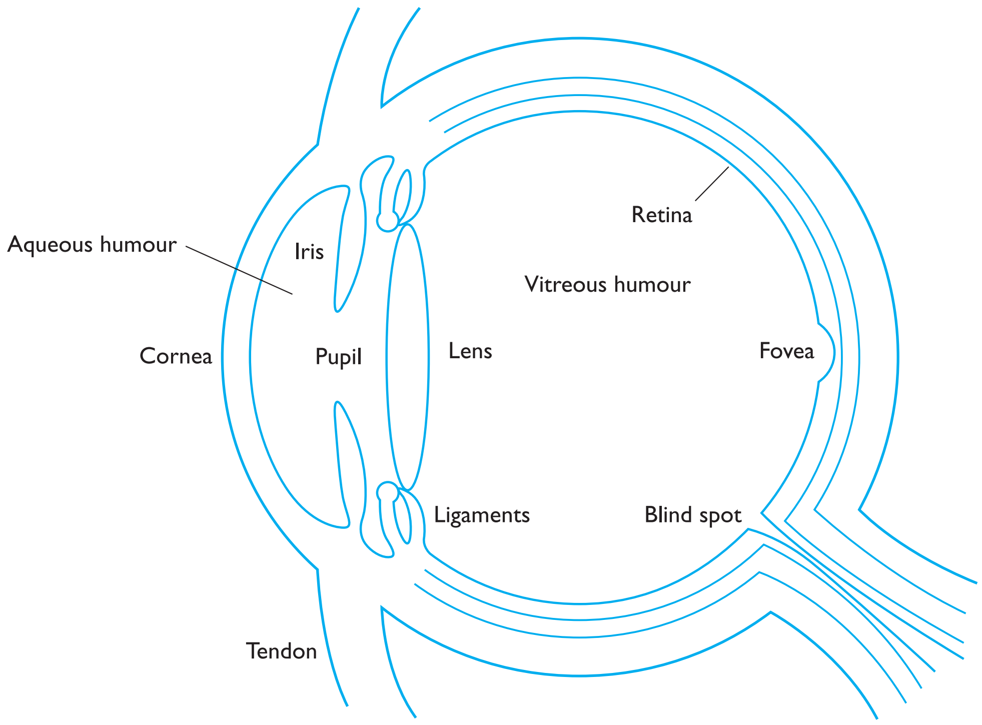


Figure 1.1 The human eye

Although the retina is mainly covered with photoreceptors there is one *blind spot* where the optic nerve enters the eye. The blind spot has no rods or cones, yet our visual system compensates for this so that in normal circumstances we are unaware of it.

The retina also has specialized nerve cells called *ganglion cells*. There are two types: X-cells, which are concentrated in the fovea and are responsible for the early detection of pattern; and Y-cells which are more widely distributed in the retina and are responsible for the early detection of movement. The distribution of these cells means that, while we may not be able to detect changes in pattern in peripheral vision, we can perceive movement.

Visual perception

Understanding the basic construction of the eye goes some way to explaining the physical mechanisms of vision but visual perception is more than this. The information received by the visual apparatus must be filtered and passed to processing elements which allow us to recognize coherent scenes, disambiguate relative distances and differentiate color. We will consider some of the capabilities and limitations of visual processing later, but first we will look a little more closely at how we perceive size and depth, brightness and color, each of which is crucial to the design of effective visual interfaces.

Design Focus

Getting noticed

The extensive knowledge about the human visual system can be brought to bear in practical design. For example, our ability to read or distinguish falls off inversely as the distance from our point of focus increases. This is due to the fact that the cones are packed more densely towards the center of our visual field. You can see this in the following image. Fixate on the dot in the center. The letters on the left should all be equally readable, those on the right all equally harder.

A B C D E F • H I J K

This loss of discrimination sets limits on the amount that can be seen or read without moving one's eyes. A user concentrating on the middle of the screen cannot be expected to read help text on the bottom line.

However, although our ability to discriminate static text diminishes, the rods, which are concentrated more in the outer parts of our visual field, are very sensitive to changes; hence we see movement well at the edge of our vision. So if you want a user to see an error message at the bottom of the screen it had better be flashing! On the other hand clever moving icons, however impressive they are, will be distracting even when the user is not looking directly at them.

Perceiving size and depth Imagine you are standing on a hilltop. Beside you on the summit you can see rocks, sheep and a small tree. On the hillside is a farmhouse with outbuildings and farm vehicles. Someone is on the track, walking toward the summit. Below in the valley is a small market town.

Even in describing such a scene the notions of size and distance predominate. Our visual system is easily able to interpret the images which it receives to take account of these things. We can identify similar objects regardless of the fact that they appear to us to be of vastly different sizes. In fact, we can use this information to judge distances.

So how does the eye perceive size, depth and relative distances? To understand this we must consider how the image appears on the retina. As we noted in the previous section, reflected light from the object forms an upside-down image on the retina. The size of that image is specified as a *visual angle*. Figure 1.2 illustrates how the visual angle is calculated.

If we were to draw a line from the top of the object to a central point on the front of the eye and a second line from the bottom of the object to the same point, the visual angle of the object is the angle between these two lines. Visual angle is affected by both the size of the object and its distance from the eye. Therefore if two objects are at the same distance, the larger one will have the larger visual angle. Similarly, if two objects of the same size are placed at different distances from the eye, the

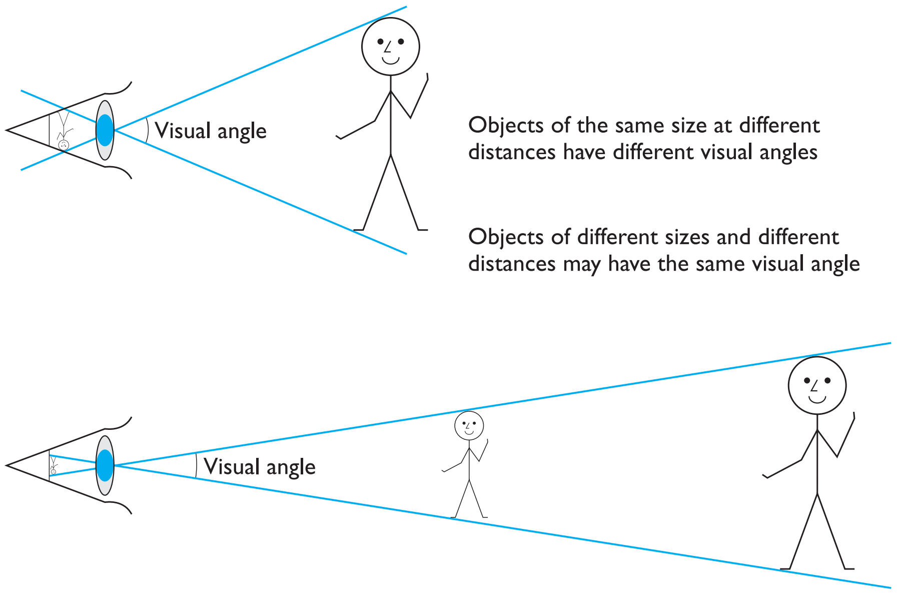

Visual angle
Objects of the same size at different distances have different visual angles
Objects of different sizes and different distances may have the same visual angle
Visual angle

Figure 1.2 Visual angle

furthest one will have the smaller visual angle. The visual angle indicates how much of the field of view is taken by the object. The visual angle measurement is given in either degrees or *minutes of arc*, where 1 degree is equivalent to 60 minutes of arc, and 1 minute of arc to 60 seconds of arc.

So how does an object's visual angle affect our perception of its size? First, if the visual angle of an object is too small we will be unable to perceive it at all. *Visual acuity* is the ability of a person to perceive fine detail. A number of measurements have been established to test visual acuity, most of which are included in standard eye tests. For example, a person with normal vision can detect a single line if it has a visual angle of 0.5 seconds of arc. Spaces between lines can be detected at 30 seconds to 1 minute of visual arc. These represent the limits of human visual acuity.

Assuming that we can perceive the object, does its visual angle affect our perception of its size? Given that the visual angle of an object is reduced as it gets further away, we might expect that we would perceive the object as smaller. In fact, our perception of an object's size remains constant even if its visual angle changes. So a person's height is perceived as constant even if they move further from you. This is the *law of size constancy*, and it indicates that our perception of size relies on factors other than the visual angle.

One of these factors is our perception of depth. If we return to the hilltop scene there are a number of *cues* which we can use to determine the relative positions and distances of the objects which we see. If objects overlap, the object which is partially covered is perceived to be in the background, and therefore further away. Similarly, the size and height of the object in our field of view provides a cue to its distance.

A third cue is familiarity: if we expect an object to be of a certain size then we can judge its distance accordingly. This has been exploited for humour in advertising: one advertisement for beer shows a man walking away from a bottle in the foreground. As he walks, he bumps into the bottle, which is in fact a giant one in the background!

Perceiving brightness A second aspect of visual perception is the perception of *brightness*. Brightness is in fact a subjective reaction to levels of light. It is affected by *luminance* which is the amount of light emitted by an object. The luminance of an object is dependent on the amount of light falling on the object's surface and its reflective properties. Luminance is a physical characteristic and can be measured using a *photometer*. *Contrast* is related to luminance: it is a function of the luminance of an object and the luminance of its background.

Although brightness is a subjective response, it can be described in terms of the amount of luminance that gives a *just noticeable difference* in brightness. However, the visual system itself also compensates for changes in brightness. In dim lighting, the rods predominate vision. Since there are fewer rods on the fovea, objects in low lighting can be seen less easily when fixated upon, and are more visible in peripheral vision. In normal lighting, the cones take over.

Visual acuity increases with increased luminance. This may be an argument for using high display luminance. However, as luminance increases, *flicker* also increases. The eye will perceive a light switched on and off rapidly as constantly on. But if the speed of switching is less than 50 Hz then the light is perceived to flicker. In high luminance flicker can be perceived at over 50 Hz. Flicker is also more noticeable in peripheral vision. This means that the larger the display (and consequently the more peripheral vision that it occupies), the more it will appear to flicker.

Perceiving color A third factor that we need to consider is perception of color. Color is usually regarded as being made up of three components: *hue*, *intensity* and *saturation*. Hue is determined by the spectral wavelength of the light. Blues have short wavelengths, greens medium and reds long. Approximately 150 different hues can be discriminated by the average person. Intensity is the brightness of the color, and saturation is the amount of whiteness in the color. By varying these two, we can perceive in the region of 7 million different colors. However, the number of colors that can be identified by an individual without training is far fewer (in the region of 10).

The eye perceives color because the cones are sensitive to light of different wavelengths. There are three different types of cone, each sensitive to a different color (blue, green and red). Color vision is best in the fovea, and worst at the periphery where rods predominate. It should also be noted that only 3–4% of the fovea is occupied by cones which are sensitive to blue light, making blue acuity lower.

Finally, we should remember that around 8% of males and 1% of females suffer from color blindness, most commonly being unable to discriminate between red and green.

The capabilities and limitations of visual processing

In considering the way in which we perceive images we have already encountered some of the capabilities and limitations of the human visual processing system. However, we have concentrated largely on low-level perception. Visual processing involves the transformation and interpretation of a complete image, from the light that is thrown onto the retina. As we have already noted, our expectations affect the way an image is perceived. For example, if we know that an object is a particular size, we will perceive it as that size no matter how far it is from us.

Visual processing compensates for the movement of the image on the retina which occurs as we move around and as the object which we see moves. Although the retinal image is moving, the image that we perceive is stable. Similarly, color and brightness of objects are perceived as constant, in spite of changes in luminance.

This ability to interpret and exploit our expectations can be used to resolve ambiguity. For example, consider the image shown in Figure 1.3. What do you perceive? Now consider Figure 1.4 and Figure 1.5. The context in which the object appears

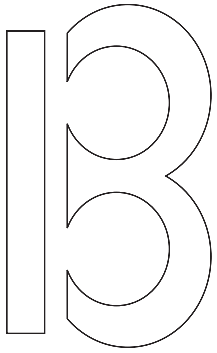


Figure 1.3 An ambiguous shape?

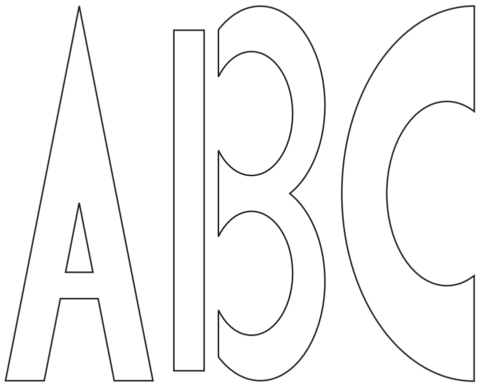


Figure 1.4 ABC

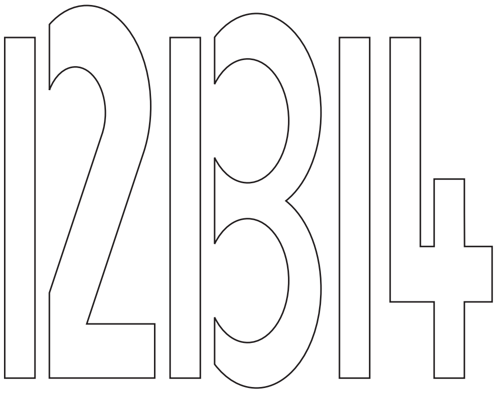


Figure 1.5 12 13 14

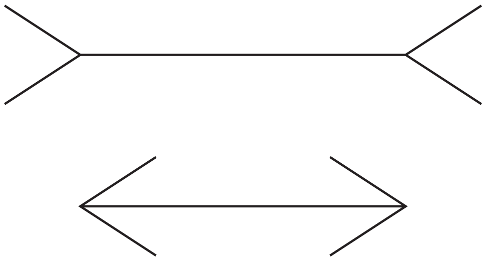


Figure 1.6 The Muller–Lyer illusion – which line is longer?

allows our expectations to clearly disambiguate the interpretation of the object, as either a B or a 13.

However, it can also create optical illusions. For example, consider Figure 1.6. Which line is longer? Most people when presented with this will say that the top line is longer than the bottom. In fact, the two lines are the same length. This may be due to a false application of the law of size constancy: the top line appears like a concave edge, the bottom like a convex edge. The former therefore seems further away than the latter and is therefore scaled to appear larger. A similar illusion is the Ponzo illusion (Figure 1.7). Here the top line appears longer, owing to the distance effect, although both lines are the same length. These illusions demonstrate that our perception of size is not completely reliable.

Another illusion created by our expectations compensating an image is the proofreading illusion. Read the text in Figure 1.8 quickly. What does it say? Most people reading this rapidly will read it correctly, although closer inspection shows that the word ‘the’ is repeated in the second and third line.

These are just a few examples of how the visual system compensates, and sometimes overcompensates, to allow us to perceive the world around us.

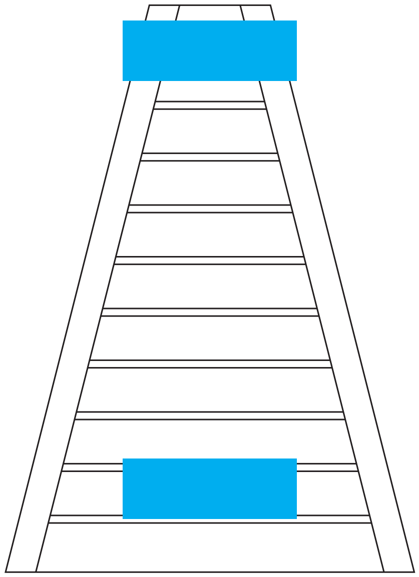


Figure 1.7 The Ponzo illusion – are these the same size?

The quick brown

fox jumps over the

the lazy dog.

Figure 1.8 Is this text correct?

DESIGN FOCUS

Where's the middle?

Optical illusions highlight the differences between the way things are and the way we perceive them – and in interface design we need to be aware that we will not always perceive things exactly as they are. The way that objects are composed together will affect the way we perceive them, and we do not perceive geometric shapes exactly as they are drawn. For example, we tend to magnify horizontal lines and reduce vertical. So a square needs to be slightly increased in height to appear square and lines will appear thicker if horizontal rather than vertical.

Optical illusions also affect page symmetry. We tend to see the center of a page as being a little above the actual center – so if a page is arranged symmetrically around the actual center, we will see it as too low down. In graphic design this is known as the *optical center* – and bottom page margins tend to be increased by 50% to compensate.

Reading

So far we have concentrated on the perception of images in general. However, the perception and processing of text is a special case that is important to interface design, which invariably requires some textual display. We will therefore end this section by looking at *reading*. There are several stages in the reading process. First, the visual pattern of the word on the page is perceived. It is then decoded with reference to an internal representation of language. The final stages of language processing include syntactic and semantic analysis and operate on phrases or sentences.

We are most concerned with the first two stages of this process and how they influence interface design. During reading, the eye makes jerky movements called *saccades* followed by fixations. Perception occurs during the fixation periods, which account for approximately 94% of the time elapsed. The eye moves backwards over the text as well as forwards, in what are known as *regressions*. If the text is complex there will be more regressions.

Adults read approximately 250 words a minute. It is unlikely that words are scanned serially, character by character, since experiments have shown that words can be recognized as quickly as single characters. Instead, familiar words are recognized using word shape. This means that removing the word shape clues (for example, by capitalizing words) is detrimental to reading speed and accuracy.

The speed at which text can be read is a measure of its legibility. Experiments have shown that standard font sizes of 9 to 12 points are equally legible, given proportional spacing between lines [346]. Similarly line lengths of between 2.3 and 5.2 inches (58 and 132 mm) are equally legible. However, there is evidence that reading from a computer screen is slower than from a book [244]. This is thought to be due to a number of factors including a longer line length, fewer words to a page, orientation and the familiarity of the medium of the page. These factors can of course be reduced by careful design of textual interfaces.

A final word about the use of contrast in visual display: a negative contrast (dark characters on a light screen) provides higher luminance and, therefore, increased acuity, than a positive contrast. This will in turn increase legibility. However, it will also be more prone to flicker. Experimental evidence suggests that in practice negative contrast displays are preferred and result in more accurate performance [30].

1.2.2 Hearing

The sense of hearing is often considered secondary to sight, but we tend to underestimate the amount of information that we receive through our ears. Close your eyes for a moment and listen. What sounds can you hear? Where are they coming from? What is making them? As I sit at my desk I can hear cars passing on the road outside, machinery working on a site nearby, the drone of a plane overhead and bird song. But I can also tell *where* the sounds are coming from, and estimate how far away they are. So from the sounds I hear I can tell that a car is passing on a particular road near my house, and which direction it is traveling in. I know that building work is in progress in a particular location, and that a certain type of bird is perched in the tree in my garden.

The auditory system can convey a lot of information about our environment. But how does it work?

The human ear

Just as vision begins with light, hearing begins with vibrations in the air or *sound waves*. The ear receives these vibrations and transmits them, through various stages, to the auditory nerves. The ear comprises three sections, commonly known as the *outer ear*, *middle ear* and *inner ear*.

The outer ear is the visible part of the ear. It has two parts: the *pinna*, which is the structure that is attached to the sides of the head, and the *auditory canal*, along which sound waves are passed to the middle ear. The outer ear serves two purposes. First, it protects the sensitive middle ear from damage. The auditory canal contains wax which prevents dust, dirt and over-inquisitive insects reaching the middle ear. It also maintains the middle ear at a constant temperature. Secondly, the pinna and auditory canal serve to amplify some sounds.

The middle ear is a small cavity connected to the outer ear by the *tympanic membrane*, or ear drum, and to the inner ear by the *cochlea*. Within the cavity are the *ossicles*, the smallest bones in the body. Sound waves pass along the auditory canal and vibrate the ear drum which in turn vibrates the ossicles, which transmit the vibrations to the cochlea, and so into the inner ear. This ‘relay’ is required because, unlike the air-filled outer and middle ears, the inner ear is filled with a denser cochlean liquid. If passed directly from the air to the liquid, the transmission of the sound waves would be poor. By transmitting them via the ossicles the sound waves are concentrated and amplified.

The waves are passed into the liquid-filled cochlea in the inner ear. Within the cochlea are delicate hair cells or *cilia* that bend because of the vibrations in the cochlean liquid and release a chemical transmitter which causes impulses in the auditory nerve.

Processing sound

As we have seen, sound is changes or vibrations in air pressure. It has a number of characteristics which we can differentiate. *Pitch* is the frequency of the sound. A low frequency produces a low pitch, a high frequency, a high pitch. *Loudness* is proportional to the amplitude of the sound; the frequency remains constant. *Timbre* relates to the type of the sound: sounds may have the same pitch and loudness but be made by different instruments and so vary in timbre. We can also identify a sound's location, since the two ears receive slightly different sounds, owing to the time difference between the sound reaching the two ears and the reduction in intensity caused by the sound waves reflecting from the head.

The human ear can hear frequencies from about 20 Hz to 15 kHz. It can distinguish frequency changes of less than 1.5 Hz at low frequencies but is less accurate at high frequencies. Different frequencies trigger activity in neurons in different parts of the auditory system, and cause different rates of firing of nerve impulses.

The auditory system performs some filtering of the sounds received, allowing us to ignore background noise and concentrate on important information. We are selective in our hearing, as illustrated by the *cocktail party effect*, where we can pick out our name spoken across a crowded noisy room. However, if sounds are too loud, or frequencies too similar, we are unable to differentiate sound.

As we have seen, sound can convey a remarkable amount of information. It is rarely used to its potential in interface design, usually being confined to warning sounds and notifications. The exception is multimedia, which may include music, voice commentary and sound effects. However, the ear can differentiate quite subtle sound changes and can recognize familiar sounds without concentrating attention on the sound source. This suggests that sound could be used more extensively in interface design, to convey information about the system state, for example. This is discussed in more detail in Chapter 10.

Worked exercise *Suggest ideas for an interface which uses the properties of sound effectively.*

Answer You might approach this exercise by considering how sound could be added to an application with which you are familiar. Use your imagination. This is also a good subject for a literature survey (starting with the references in Chapter 10).

Speech sounds can obviously be used to convey information. This is useful not only for the visually impaired but also for any application where the user's attention has to be divided (for example, power plant control, flight control, etc.). Uses of non-speech sounds include the following:

- Attention – to attract the user's attention to a critical situation or to the end of a process, for example.

- Status information – continuous background sounds can be used to convey status information. For example, monitoring the progress of a process (without the need for visual attention).
- Confirmation – a sound associated with an action to confirm that the action has been carried out. For example, associating a sound with deleting a file.
- Navigation – using changing sound to indicate where the user is in a system. For example, what about sound to support navigation in hypertext?

1.2.3 Touch

The third and last of the senses that we will consider is touch or *haptic perception*. Although this sense is often viewed as less important than sight or hearing, imagine life without it. Touch provides us with vital information about our environment. It tells us when we touch something hot or cold, and can therefore act as a warning. It also provides us with feedback when we attempt to lift an object, for example. Consider the act of picking up a glass of water. If we could only see the glass and not feel when our hand made contact with it or feel its shape, the speed and accuracy of the action would be reduced. This is the experience of users of certain *virtual reality* games: they can see the computer-generated objects which they need to manipulate but they have no physical sensation of touching them. Watching such users can be an informative and amusing experience! Touch is therefore an important means of feedback, and this is no less so in using computer systems. Feeling buttons depress is an important part of the task of pressing the button. Also, we should be aware that, although for the average person, haptic perception is a secondary source of information, for those whose other senses are impaired, it may be vitally important. For such users, interfaces such as braille may be the primary source of information in the interaction. We should not therefore underestimate the importance of touch.

The apparatus of touch differs from that of sight and hearing in that it is not localized. We receive stimuli through the skin. The skin contains three types of sensory receptor: *thermoreceptors* respond to heat and cold, *nociceptors* respond to intense pressure, heat and pain, and *mechanoreceptors* respond to pressure. It is the last of these that we are concerned with in relation to human–computer interaction.

There are two kinds of mechanoreceptor, which respond to different types of pressure. *Rapidly adapting mechanoreceptors* respond to immediate pressure as the skin is indented. These receptors also react more quickly with increased pressure. However, they stop responding if continuous pressure is applied. *Slowly adapting mechanoreceptors* respond to continuously applied pressure.

Although the whole of the body contains such receptors, some areas have greater sensitivity or acuity than others. It is possible to measure the acuity of different areas of the body using the *two-point threshold test*. Take two pencils, held so their tips are about 12 mm apart. Touch the points to your thumb and see if you can feel two points. If you cannot, move the points a little further apart. When you can feel two points, measure the distance between them. The greater the distance, the lower the sensitivity. You can repeat this test on different parts of your body. You should find that the measure on the forearm is around 10 times that of the finger or thumb. The fingers and thumbs have the highest acuity.

A second aspect of haptic perception is *kinesthesis*: awareness of the position of the body and limbs. This is due to receptors in the joints. Again there are three types: rapidly adapting, which respond when a limb is moved in a particular direction; slowly adapting, which respond to both movement and static position; and positional receptors, which only respond when a limb is in a static position. This perception affects both comfort and performance. For example, for a touch typist, awareness of the relative positions of the fingers and feedback from the keyboard are very important.

Handling the goods

E-commerce has become very successful in some areas of sales, such as travel services, books and CDs, and food. However, in some retail areas, such as clothes shopping, e-commerce has been less successful. Why?

When buying train and airline tickets and, to some extent, books and food, the experience of shopping is less important than the convenience. So, as long as we know what we want, we are happy to shop online. With clothes, the experience of shopping is far more important. We need to be able to handle the goods, feel the texture of the material, check the weight to test quality. Even if we know that something will fit us we still want to be able to handle it before buying.

Research into haptic interaction (see Chapter 2 and Chapter 10) is looking at ways of solving this problem. By using special force feedback and tactile hardware, users are able to feel surfaces and shape. For example, a demonstration environment called TouchCity allows people to walk around a virtual shopping mall, pick up products and feel their texture and weight. A key problem with the commercial use of such an application, however, is that the haptic experience requires expensive hardware not yet available to the average e-shopper. However, in future, such immersive e-commerce experiences are likely to be the norm. (See www.novint.com/)

1.2.4 Movement

Before leaving this section on the human's input–output channels, we need to consider motor control and how the way we move affects our interaction with computers. A simple action such as hitting a button in response to a question involves a number of processing stages. The stimulus (of the question) is received through the sensory receptors and transmitted to the brain. The question is processed and a valid response generated. The brain then tells the appropriate muscles to respond. Each of these stages takes time, which can be roughly divided into reaction time and movement time.

Movement time is dependent largely on the physical characteristics of the subjects: their age and fitness, for example. Reaction time varies according to the sensory channel through which the stimulus is received. A person can react to an auditory signal in approximately 150 ms, to a visual signal in 200 ms and to pain in 700 ms. However, a combined signal will result in the quickest response. Factors such as skill or practice can reduce reaction time, and fatigue can increase it.

A second measure of motor skill is accuracy. One question that we should ask is whether speed of reaction results in reduced accuracy. This is dependent on the task and the user. In some cases, requiring increased reaction time reduces accuracy. This is the premise behind many arcade and video games where less skilled users fail at levels of play that require faster responses. However, for skilled operators this is not necessarily the case. Studies of keyboard operators have shown that, although the faster operators were up to twice as fast as the others, the slower ones made 10 times the errors.

Speed and accuracy of movement are important considerations in the design of interactive systems, primarily in terms of the time taken to move to a particular target on a screen. The target may be a button, a menu item or an icon, for example. The time taken to hit a target is a function of the size of the target and the distance that has to be moved. This is formalized in *Fitts' law* [135]. There are many variations of this formula, which have varying constants, but they are all very similar. One common form is

$$\text{Movement time} = a + b \log_2(\text{distance/size} + 1)$$

where a and b are empirically determined constants.

This affects the type of target we design. Since users will find it more difficult to manipulate small objects, targets should generally be as large as possible and the distance to be moved as small as possible. This has led to suggestions that pie-chart-shaped menus are preferable to lists since all options are equidistant. However, the trade-off is increased use of screen estate, so the choice may not be so simple. If lists are used, the most frequently used options can be placed closest to the user's start point (for example, at the top of the menu). The implications of Fitts' law in design are discussed in more detail in Chapter 12.

1.3 HUMAN MEMORY

Have you ever played the memory game? The idea is that each player has to recount a list of objects and add one more to the end. There are many variations but the objects are all loosely related: 'I went to the market and bought a lemon, some oranges, bacon . . .' or 'I went to the zoo and saw monkeys, and lions, and tigers . . .' and so on. As the list grows objects are missed out or recalled in the wrong order and so people are eliminated from the game. The winner is the person remaining at the end. Such games rely on our ability to store and retrieve information, even seemingly arbitrary items. This is the job of our memory system.

Indeed, much of our everyday activity relies on memory. As well as storing all our factual knowledge, our memory contains our knowledge of actions or procedures.

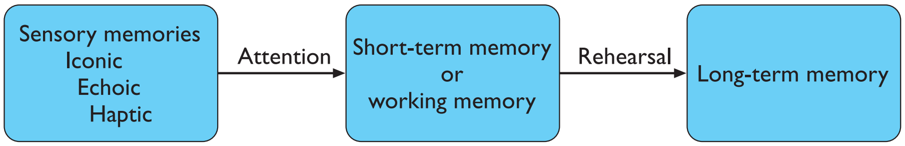


Figure 1.9 A model of the structure of memory

It allows us to repeat actions, to use language, and to use new information received via our senses. It also gives us our sense of identity, by preserving information from our past experiences.

But how does our memory work? How do we remember arbitrary lists such as those generated in the memory game? Why do some people remember more easily than others? And what happens when we forget?

In order to answer questions such as these, we need to understand some of the capabilities and limitations of human memory. Memory is the second part of our model of the human as an information-processing system. However, as we noted earlier, such a division is simplistic since, as we shall see, memory is associated with each level of processing. Bearing this in mind, we will consider the way in which memory is structured and the activities that take place within the system.

It is generally agreed that there are three types of memory or memory function: *sensory buffers*, *short-term memory* or *working memory*, and *long-term memory*. There is some disagreement as to whether these are three separate systems or different functions of the same system. We will not concern ourselves here with the details of this debate, which is discussed in detail by Baddeley [21], but will indicate the evidence used by both sides as we go along. For our purposes, it is sufficient to note three separate types of memory. These memories interact, with information being processed and passed between memory stores, as shown in Figure 1.9.

1.3.1 Sensory memory

The sensory memories act as buffers for stimuli received through the senses. A sensory memory exists for each sensory channel: *iconic memory* for visual stimuli, *echoic memory* for aural stimuli and *haptic memory* for touch. These memories are constantly overwritten by new information coming in on these channels.

We can demonstrate the existence of iconic memory by moving a finger in front of the eye. Can you see it in more than one place at once? This indicates a persistence of the image after the stimulus has been removed. A similar effect is noticed most vividly at firework displays where moving sparklers leave a persistent image. Information remains in iconic memory very briefly, in the order of 0.5 seconds.

Similarly, the existence of echoic memory is evidenced by our ability to ascertain the direction from which a sound originates. This is due to information being received by both ears. However, since this information is received at different times, we must store the stimulus in the meantime. Echoic memory allows brief 'play-back' of information. Have you ever had someone ask you a question when you are reading? You ask them to repeat the question, only to realize that you know what was asked after all. This experience, too, is evidence of the existence of echoic memory.

Information is passed from sensory memory into short-term memory by attention, thereby filtering the stimuli to only those which are of interest at a given time. Attention is the concentration of the mind on one out of a number of competing stimuli or thoughts. It is clear that we are able to focus our attention selectively, choosing to attend to one thing rather than another. This is due to the limited capacity of our sensory and mental processes. If we did not selectively attend to the stimuli coming into our senses, we would be overloaded. We can choose which stimuli to attend to, and this choice is governed to an extent by our *arousal*, our level of interest or need. This explains the cocktail party phenomenon mentioned earlier: we can attend to one conversation over the background noise, but we may choose to switch our attention to a conversation across the room if we hear our name mentioned. Information received by sensory memories is quickly passed into a more permanent memory store, or overwritten and lost.

1.3.2 Short-term memory

Short-term memory or working memory acts as a 'scratch-pad' for temporary recall of information. It is used to store information which is only required fleetingly. For example, calculate the multiplication 35×6 in your head. The chances are that you will have done this calculation in stages, perhaps 5×6 and then 30×6 and added the results; or you may have used the fact that $6 = 2 \times 3$ and calculated $2 \times 35 = 70$ followed by 3×70. To perform calculations such as this we need to store the intermediate stages for use later. Or consider reading. In order to comprehend this sentence you need to hold in your mind the beginning of the sentence as you read the rest. Both of these tasks use short-term memory.

Short-term memory can be accessed rapidly, in the order of 70 ms. However, it also decays rapidly, meaning that information can only be held there temporarily, in the order of 200 ms.

Short-term memory also has a limited capacity. There are two basic methods for measuring memory capacity. The first involves determining the length of a sequence which can be remembered in order. The second allows items to be freely recalled in any order. Using the first measure, the average person can remember 7 ± 2 digits. This was established in experiments by Miller [234]. Try it. Look at the following number sequence:

265397620853

Now write down as much of the sequence as you can remember. Did you get it all right? If not, how many digits could you remember? If you remembered between five and nine digits your *digit span* is average.

Now try the following sequence:

44 113 245 8920

Did you recall that more easily? Here the digits are grouped or *chunked*. A generalization of the 7 ± 2 rule is that we can remember 7 ± 2 *chunks* of information. Therefore chunking information can increase the short-term memory capacity. The limited capacity of short-term memory produces a subconscious desire to create chunks, and so optimize the use of the memory. The successful formation of a chunk is known as *closure*. This process can be generalized to account for the desire to complete or close tasks held in short-term memory. If a subject fails to do this or is prevented from doing so by interference, the subject is liable to lose track of what she is doing and make consequent errors.

Design Focus

Cashing in

Closure gives you a nice 'done it' when we complete some part of a task. At this point our minds have a tendency to flush short-term memory in order to get on with the next job. Early automatic teller machines (ATMs) gave the customer money before returning their bank card. On receiving the money the customer would reach closure and hence often forget to take the card. Modern ATMs return the card first!

The sequence of chunks given above also makes use of pattern abstraction: it is written in the form of a UK telephone number which makes it easier to remember. We may even recognize the first sets of digits as the international code for the UK and the dialing code for Leeds – chunks of information. Patterns can be useful as aids to memory. For example, most people would have difficulty remembering the following sequence of chunks:

HEC ATR ANU PTH ETR EET

However, if you notice that by moving the last character to the first position, you get the statement 'the cat ran up the tree', the sequence is easy to recall.

In experiments where subjects were able to recall words freely, evidence shows that recall of the last words presented is better than recall of those in the middle [296]. This is known as the *recency effect*. However, if the subject is asked to perform another task between presentation and recall (for example, counting backwards) the recency effect is eliminated. The recall of the other words is unaffected. This suggests that short-term memory recall is damaged by interference of other information. However, the fact that this interference does not affect recall of earlier items provides some evidence for the existence of separate long-term and short-term memories. The early items are held in a long-term store which is unaffected by the recency effect.

Interference does not necessarily impair recall in short-term memory. Baddeley asked subjects to remember six-digit numbers and attend to sentence processing at the same time [21]. They were asked to answer questions on sentences, such as 'A precedes B: AB is true or false?'. Surprisingly, this did not result in interference, suggesting that in fact short-term memory is not a unitary system but is made up of a number of components, including a visual channel and an articulatory channel. The task of sentence processing used the visual channel, while the task of remembering digits used the articulatory channel, so interference only occurs if tasks utilize the same channel.

These findings led Baddeley to propose a model of working memory that incorporated a number of elements together with a central processing executive. This is illustrated in Figure 1.10.

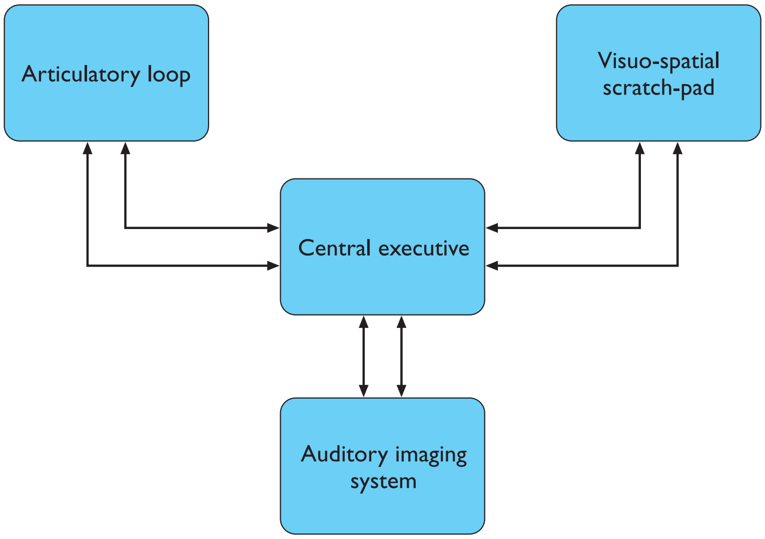


Figure 1.10 A more detailed model of short-term memory

Design Focus

7 ± 2 revisited

When we looked at short-term memory, we noted the general rule that people can hold 7 ± 2 items or chunks of information in short-term memory. It is a principle that people tend to remember but it can be misapplied. For example, it is often suggested that this means that lists, menus and other groups of items should be designed to be no more than 7 items long. But use of menus and lists of course has little to do with short-term memory – they are available in the environment as cues and so do not need to be remembered.

On the other hand the 7 ± 2 rule would apply in command line interfaces. Imagine a scenario where a UNIX user looks up a command in the manual. Perhaps the command has a number of parameters of options, to be applied in a particular order, and it is going to be applied to several files that have long path names. The user then has to hold the command, its parameters and the file path names in short-term memory while he types them in. Here we could say that the task may cause problems if the number of items or chunks in the command line string is more than 7.

1.3.3 Long-term memory

If short-term memory is our working memory or ‘scratch-pad’, long-term memory is our main resource. Here we store factual information, experiential knowledge, procedural rules of behavior – in fact, everything that we ‘know’. It differs from short-term memory in a number of significant ways. First, it has a huge, if not unlimited, capacity. Secondly, it has a relatively slow access time of approximately a tenth of a second. Thirdly, forgetting occurs more slowly in long-term memory, if at all. These distinctions provide further evidence of a memory structure with several parts.

Long-term memory is intended for the long-term storage of information. Information is placed there from working memory through rehearsal. Unlike working memory there is little decay: long-term recall after minutes is the same as that after hours or days.

Long-term memory structure

There are two types of long-term memory: *episodic memory* and *semantic memory*. Episodic memory represents our memory of events and experiences in a serial form. It is from this memory that we can reconstruct the actual events that took place at a given point in our lives. Semantic memory, on the other hand, is a structured record of facts, concepts and skills that we have acquired. The information in semantic memory is derived from that in our episodic memory, such that we can learn new facts or concepts from our experiences.

Semantic memory is structured in some way to allow access to information, representation of relationships between pieces of information, and inference. One model for the way in which semantic memory is structured is as a network. Items are

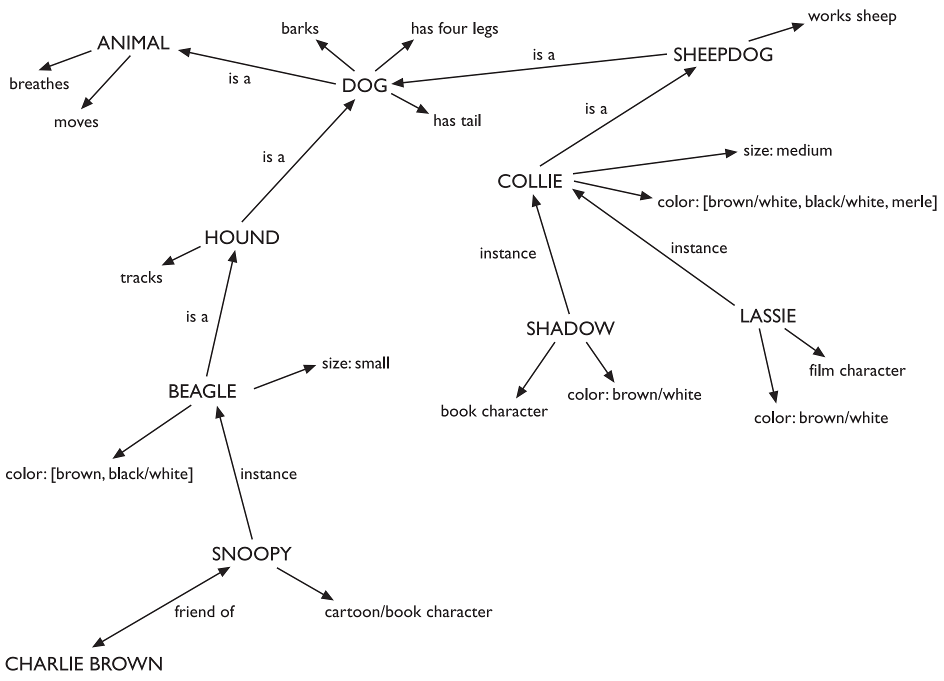


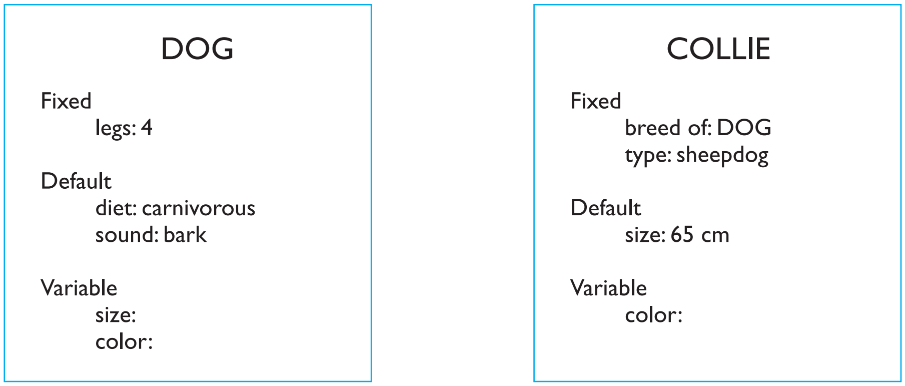
Figure 1.11 Long-term memory may store information in a semantic network

associated to each other in classes, and may inherit attributes from parent classes. This model is known as a *semantic network*. As an example, our knowledge about dogs may be stored in a network such as that shown in Figure 1.11.

Specific breed attributes may be stored with each given breed, yet general dog information is stored at a higher level. This allows us to generalize about specific cases. For instance, we may not have been told that the sheepdog Shadow has four legs and a tail, but we can infer this information from our general knowledge about sheepdogs and dogs in general. Note also that there are connections within the network which link into other domains of knowledge, for example cartoon characters. This illustrates how our knowledge is organized by association.

The viability of semantic networks as a model of memory organization has been demonstrated by Collins and Quillian [74]. Subjects were asked questions about different properties of related objects and their reaction times were measured. The types of question asked (taking examples from our own network) were 'Can a collie breathe?', 'Is a beagle a hound?' and 'Does a hound track?' In spite of the fact that the answers to such questions may seem obvious, subjects took longer to answer questions such as 'Can a collie breathe?' than ones such as 'Does a hound track?' The reason for this, it is suggested, is that in the former case subjects had to search further through the memory hierarchy to find the answer, since information is stored at its most abstract level.

A number of other memory structures have been proposed to explain how we represent and store different types of knowledge. Each of these represents a different



DOG

Fixed
- legs: 4

Default
- diet: carnivorous
- sound: bark

Variable
- size:
- color:

COLLIE

Fixed
- breed of: DOG
- type: sheepdog

Default
- size: 65 cm

Variable
- color:

Figure 1.12 A frame-based representation of knowledge

aspect of knowledge and, as such, the models can be viewed as complementary rather than mutually exclusive. Semantic networks represent the associations and relationships between single items in memory. However, they do not allow us to model the representation of more complex objects or events, which are perhaps composed of a number of items or activities. Structured representations such as *frames* and *scripts* organize information into data structures. *Slots* in these structures allow attribute values to be added. Frame slots may contain default, fixed or variable information. A frame is instantiated when the slots are filled with appropriate values. Frames and scripts can be linked together in networks to represent hierarchical structured knowledge.

Returning to the 'dog' domain, a frame-based representation of the knowledge may look something like Figure 1.12. The fixed slots are those for which the attribute value is set, default slots represent the usual attribute value, although this may be overridden in particular instantiations (for example, the Basenji does not bark), and variable slots can be filled with particular values in a given instance. Slots can also contain procedural knowledge. Actions or operations can be associated with a slot and performed, for example, whenever the value of the slot is changed.

Frames extend semantic nets to include structured, hierarchical information. They represent knowledge items in a way which makes explicit the relative importance of each piece of information.

Scripts attempt to model the representation of stereotypical knowledge about situations. Consider the following sentence:

John took his dog to the surgery. After seeing the vet, he left.

From our knowledge of the activities of dog owners and vets, we may fill in a substantial amount of detail. The animal was ill. The vet examined and treated the animal. John paid for the treatment before leaving. We are less likely to assume the alternative reading of the sentence, that John took an instant dislike to the vet on sight and did not stay long enough to talk to him!

Script for a visit to the vet

Entry conditions:	*dog ill* *vet open* *owner has money*	Roles:	*vet examines* *diagnoses* *treats* *owner brings dog in* *pays* *takes dog out*
Result:	*dog better* *owner poorer* *vet richer*	Scenes:	*arriving at reception* *waiting in room* *examination* *paying*
Props:	*examination table* *medicine* *instruments*	Tracks:	*dog needs medicine* *dog needs operation*

Figure 1.13 A script for visiting the vet

A script represents this default or stereotypical information, allowing us to interpret partial descriptions or cues fully. A script comprises a number of elements, which, like slots, can be filled with appropriate information:

Entry conditions Conditions that must be satisfied for the script to be activated.

Result Conditions that will be true after the script is terminated.

Props Objects involved in the events described in the script.

Roles Actions performed by particular participants.

Scenes The sequences of events that occur.

Tracks A variation on the general pattern representing an alternative scenario.

An example script for going to the vet is shown in Figure 1.13.

A final type of knowledge representation which we hold in memory is the representation of procedural knowledge, our knowledge of how to do something. A common model for this is the production system. Condition–action rules are stored in long-term memory. Information coming into short-term memory can match a condition in one of these rules and result in the action being executed. For example, a pair of production rules might be

```
IF dog is wagging tail
THEN pat dog

IF dog is growling
THEN run away
```

If we then meet a growling dog, the condition in the second rule is matched, and we respond by turning tail and running. (Not to be recommended by the way!)

Long-term memory processes

So much for the structure of memory, but what about the processes which it uses? There are three main activities related to long-term memory: storage or remembering of information, forgetting and information retrieval. We shall consider each of these in turn.

First, how does information get into long-term memory and how can we improve this process? Information from short-term memory is stored in long-term memory by rehearsal. The repeated exposure to a stimulus or the rehearsal of a piece of information transfers it into long-term memory.

This process can be optimized in a number of ways. Ebbinghaus performed numerous experiments on memory, using himself as a subject [117]. In these experiments he tested his ability to learn and repeat nonsense syllables, comparing his recall minutes, hours and days after the learning process. He discovered that the amount learned was directly proportional to the amount of time spent learning. This is known as the *total time hypothesis*. However, experiments by Baddeley and others suggest that learning time is most effective if it is distributed over time [22]. For example, in an experiment in which Post Office workers were taught to type, those whose training period was divided into weekly sessions of one hour performed better than those who spent two or four hours a week learning (although the former obviously took more weeks to complete their training). This is known as the *distribution of practice effect*.

However, repetition is not enough to learn information well. If information is not meaningful it is more difficult to remember. This is illustrated by the fact that it is more difficult to remember a set of words representing concepts than a set of words representing objects. Try it. First try to remember the words in list A and test yourself.

List A: Faith Age Cold Tenet Quiet Logic Idea Value Past Large

Now try list B.

List B: Boat Tree Cat Child Rug Plate Church Gun Flame Head

The second list was probably easier to remember than the first since you could visualize the objects in the second list.

Sentences are easier still to memorize. Bartlett performed experiments on remembering meaningful information (as opposed to meaningless such as Ebbinghaus used) [28]. In one such experiment he got subjects to learn a story about an unfamiliar culture and then retell it. He found that subjects would retell the story replacing unfamiliar words and concepts with words which were meaningful to them. Stories were effectively translated into the subject's own culture. This is related to the semantic structuring of long-term memory: if information is meaningful and familiar, it can be related to existing structures and more easily incorporated into memory.

Memorable or secure?

As online activities become more widespread, people are having to remember more and more access information, such as passwords and security checks. The average active internet user may have separate passwords and user names for several email accounts, mailing lists, e-shopping sites, e-banking, online auctions and more! Remembering these passwords is not easy.

From a security perspective it is important that passwords are random. Words and names are very easy to crack, hence the recommendation that passwords are frequently changed and constructed from random strings of letters and numbers. But in reality these are the hardest things for people to commit to memory. Hence many people will use the same password for all their online activities (rarely if ever changing it) and will choose a word or a name that is easy for them to remember, in spite of the obviously increased security risks. Security here is in conflict with memorability!

A solution to this is to construct a nonsense password out of letters or numbers that will have meaning to you but will not make up a word in a dictionary (e.g. initials of names, numbers from significant dates or postcodes, and so on). Then what is remembered is the meaningful rule for constructing the password, and not a meaningless string of alphanumeric characters.

So if structure, familiarity and concreteness help us in learning information, what causes us to lose this information, to forget? There are two main theories of forgetting: *decay* and *interference*. The first theory suggests that the information held in long-term memory may eventually be forgotten. Ebbinghaus concluded from his experiments with nonsense syllables that information in memory decayed logarithmically, that is that it was lost rapidly to begin with, and then more slowly. *Jost's law*, which follows from this, states that if two memory traces are equally strong at a given time the older one will be more durable.

The second theory is that information is lost from memory through interference. If we acquire new information it causes the loss of old information. This is termed *retroactive interference*. A common example of this is the fact that if you change telephone numbers, learning your new number makes it more difficult to remember your old number. This is because the new association masks the old. However, sometimes the old memory trace breaks through and interferes with new information. This is called *proactive inhibition*. An example of this is when you find yourself driving to your old house rather than your new one.

Forgetting is also affected by emotional factors. In experiments, subjects given emotive words and non-emotive words found the former harder to remember in the short term but easier in the long term. Indeed, this observation tallies with our experience of selective memory. We tend to remember positive information rather than negative (hence nostalgia for the 'good old days'), and highly emotive events rather than mundane.

It is debatable whether we ever actually forget anything or whether it just becomes increasingly difficult to access certain items from memory. This question is in some ways moot since it is impossible to prove that we *do* forget: appearing to have forgotten something may just be caused by not being able to retrieve it! However, there is evidence to suggest that we may not lose information completely from long-term memory. First, proactive inhibition demonstrates the recovery of old information even after it has been 'lost' by interference. Secondly, there is the 'tip of the tongue' experience, which indicates that some information is present but cannot be satisfactorily accessed. Thirdly, information may not be recalled but may be recognized, or may be recalled only with prompting.

This leads us to the third process of memory: information retrieval. Here we need to distinguish between two types of information retrieval, recall and recognition. In recall the information is reproduced from memory. In recognition, the presentation of the information provides the knowledge that the information has been seen before. Recognition is the less complex cognitive activity since the information is provided as a cue.

However, recall can be assisted by the provision of retrieval cues, which enable the subject quickly to access the information in memory. One such cue is the use of categories. In an experiment subjects were asked to recall lists of words, some of which were organized into categories and some of which were randomly organized. The words that were related to a category were easier to recall than the others [38]. Recall is even more successful if subjects are allowed to categorize their own lists of words during learning. For example, consider the following list of words:

child red plane dog friend blood cold tree big angry

Now make up a story that links the words using as vivid imagery as possible. Now try to recall as many of the words as you can. Did you find this easier than the previous experiment where the words were unrelated?

The use of vivid imagery is a common cue to help people remember information. It is known that people often visualize a scene that is described to them. They can then answer questions based on their visualization. Indeed, subjects given a description of a scene often embellish it with additional information. Consider the following description and imagine the scene:

> The engines roared above the noise of the crowd. Even in the blistering heat people rose to their feet and waved their hands in excitement. The flag fell and they were off. Within seconds the car had pulled away from the pack and was careering round the bend at a desperate pace. Its wheels momentarily left the ground as it cornered. Coming down the straight the sun glinted on its shimmering paint. The driver gripped the wheel with fierce concentration. Sweat lay in fine drops on his brow.

Without looking back to the passage, what color is the car?

If you could answer that question you have visualized the scene, including the car's color. In fact, the color of the car is not mentioned in the description at all.

Improve your memory

Many people can perform astonishing feats of memory: recalling the sequence of cards in a pack (or multiple packs – up to six have been reported), or recounting π to 1000 decimal places, for example. There are also adverts to 'Improve Your Memory' (usually leading to success, or wealth, or other such inducement), and so the question arises: can you improve your memory abilities? The answer is yes; this exercise shows you one technique.

Look at the list below of numbers and associated words:

1	bun	6	sticks
2	shoe	7	heaven
3	tree	8	gate
4	door	9	wine
5	hive	10	hen

Notice that the words sound similar to the numbers. Now think about the words one at a time and visualize them, in as much detail as possible. For example, for '1', think of a large, sticky iced bun, the base spiralling round and round, with raisins in it, covered in sweet, white, gooey icing. Now do the rest, using as much visualization as you can muster: imagine how things would look, smell, taste, sound, and so on.

This is your reference list, and you need to know it off by heart.

Having learnt it, look at a pile of at least a dozen odd items collected together by a colleague. The task is to look at the collection of objects for only 30 seconds, and then list as many as possible without making a mistake or viewing the collection again. Most people can manage between five and eight items, if they do not know any memory-enhancing techniques like the following.

Mentally pick one (say, for example, a paper clip), and call it number one. Now visualize it interacting with the bun. It can get stuck into the icing on the top of the bun, and make your fingers all gooey and sticky when you try to remove it. If you ate the bun without noticing, you'd get a crunched tooth when you bit into it – imagine how that would feel. When you've really got a graphic scenario developed, move on to the next item, call it number two, and again visualize it interacting with the reference item, shoe. Continue down your list, until you have done 10 things.

This should take you about the 30 seconds allowed. Then hide the collection and try and recall the numbers in order, the associated reference word, and then the image associated with that word. You should find that you can recall the 10 associated items practically every time. The technique can be easily extended by extending your reference list.

1.4 THINKING: REASONING AND PROBLEM SOLVING

We have considered how information finds its way into and out of the human system and how it is stored. Finally, we come to look at how it is processed and manipulated. This is perhaps the area which is most complex and which separates humans from other information-processing systems, both artificial and natural. Although it is clear that animals receive and store information, there is little evidence to suggest that they can use it in quite the same way as humans. Similarly, artificial intelligence has produced machines which can see (albeit in a limited way) and store information. But their ability to use that information is limited to small domains.

Humans, on the other hand, are able to use information to reason and solve problems, and indeed do these activities when the information is partial or unavailable. Human thought is conscious and self-aware: while we may not always be able to identify the processes we use, we can identify the products of these processes, our thoughts. In addition, we are able to think about things of which we have no experience, and solve problems which we have never seen before. How is this done?

Thinking can require different amounts of knowledge. Some thinking activities are very directed and the knowledge required is constrained. Others require vast amounts of knowledge from different domains. For example, performing a subtraction calculation requires a relatively small amount of knowledge, from a constrained domain, whereas understanding newspaper headlines demands knowledge of politics, social structures, public figures and world events.

In this section we will consider two categories of thinking: reasoning and problem solving. In practice these are not distinct since the activity of solving a problem may well involve reasoning and vice versa. However, the distinction is a common one and is helpful in clarifying the processes involved.

1.4.1 Reasoning

Reasoning is the process by which we use the knowledge we have to draw conclusions or infer something new about the domain of interest. There are a number of different types of reasoning: *deductive*, *inductive* and *abductive*. We use each of these types of reasoning in everyday life, but they differ in significant ways.

Deductive reasoning

Deductive reasoning derives the logically necessary conclusion from the given premises. For example,

> If it is Friday then she will go to work
> It is Friday
> Therefore she will go to work.

It is important to note that this is the *logical* conclusion from the premises; it does not necessarily have to correspond to our notion of truth. So, for example,

> If it is raining then the ground is dry
> It is raining
> Therefore the ground is dry.

is a perfectly valid deduction, even though it conflicts with our knowledge of what is true in the world.

Deductive reasoning is therefore often misapplied. Given the premises

Some people are babies
Some babies cry

many people will infer that 'Some people cry'. This is in fact an invalid deduction since we are not told that all babies are people. It is therefore logically possible that the babies who cry are those who are not people.

It is at this point, where truth and validity clash, that human deduction is poorest. One explanation for this is that people bring their world knowledge into the reasoning process. There is good reason for this. It allows us to take short cuts which make dialog and interaction between people informative but efficient. We assume a certain amount of shared knowledge in our dealings with each other, which in turn allows us to interpret the inferences and deductions implied by others. If validity rather than truth was preferred, all premises would have to be made explicit.

Inductive reasoning

Induction is generalizing from cases we have seen to infer information about cases we have not seen. For example, if every elephant we have ever seen has a trunk, we infer that all elephants have trunks. Of course, this inference is unreliable and cannot be proved to be true; it can only be proved to be false. We can disprove the inference simply by producing an elephant without a trunk. However, we can never prove it true because, no matter how many elephants with trunks we have seen or are known to exist, the next one we see may be trunkless. The best that we can do is gather evidence to support our inductive inference.

In spite of its unreliability, induction is a useful process, which we use constantly in learning about our environment. We can never see all the elephants that have ever lived or will ever live, but we have certain knowledge about elephants which we are prepared to trust for all practical purposes, which has largely been inferred by induction. Even if we saw an elephant without a trunk, we would be unlikely to move from our position that 'All elephants have trunks', since we are better at using positive than negative evidence. This is illustrated in an experiment first devised by Wason [365]. You are presented with four cards as in Figure 1.14. Each card has a number on one side and a letter on the other. Which cards would you need to pick up to test the truth of the statement 'If a card has a vowel on one side it has an even number on the other'?

A common response to this (was it yours?) is to check the E and the 4. However, this uses only positive evidence. In fact, to test the truth of the statement we need to check negative evidence: if we can find a card which has an odd number on one side and a vowel on the other we have disproved the statement. We must therefore check E and 7. (It does not matter what is on the other side of the other cards: the statement does not say that all even numbers have vowels, just that all vowels have even numbers.)

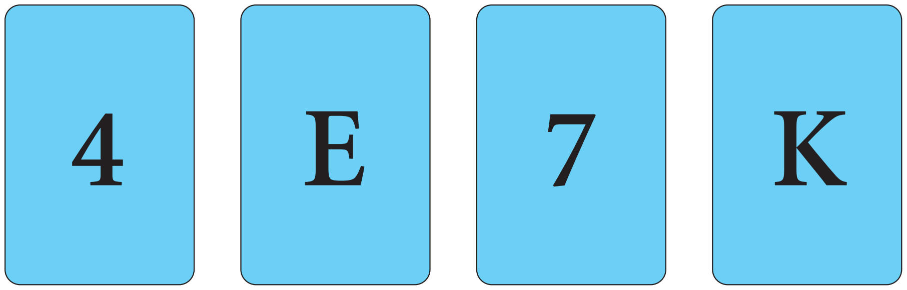


Figure 1.14 Wason's cards

Filling the gaps

Look again at Wason's cards in Figure 1.14. In the text we say that you only need to check the E and the 7. This is correct, but only because we very carefully stated in the text that 'each card has a number on one side and a letter on the other'. If the problem were stated without that condition then the K would also need to be examined in case it has a vowel on the other side. In fact, when the problem is so stated, even the most careful subjects ignore this possibility. Why? Because the nature of the problem implicitly suggests that each card has a number on one side and a letter on the other.

This is similar to the embellishment of the story at the end of Section 1.3.3. In fact, we constantly fill in gaps in the evidence that reaches us through our senses. Although this can lead to errors in our reasoning it is also essential for us to function. In the real world we rarely have all the evidence necessary for logical deductions and at all levels of perception and reasoning we fill in details in order to allow higher levels of reasoning to work.

Abductive reasoning

The third type of reasoning is abduction. Abduction reasons from a fact to the action or state that caused it. This is the method we use to derive explanations for the events we observe. For example, suppose we know that Sam always drives too fast when she has been drinking. If we see Sam driving too fast we may infer that she has been drinking. Of course, this too is unreliable since there may be another reason why she is driving fast: she may have been called to an emergency, for example.

In spite of its unreliability, it is clear that people do infer explanations in this way, and hold onto them until they have evidence to support an alternative theory or explanation. This can lead to problems in using interactive systems. If an event always follows an action, the user will infer that the event is caused by the action unless evidence to the contrary is made available. If, in fact, the event and the action are unrelated, confusion and even error often result.

1.4.2 Problem solving

If reasoning is a means of inferring new information from what is already known, problem solving is the process of finding a solution to an unfamiliar task, using the knowledge we have. Human problem solving is characterized by the ability to adapt the information we have to deal with new situations. However, often solutions seem to be original and creative. There are a number of different views of how people solve problems. The earliest, dating back to the first half of the twentieth century, is the *Gestalt* view that problem solving involves both reuse of knowledge and insight. This has been largely superseded but the questions it was trying to address remain and its influence can be seen in later research. A second major theory, proposed in the 1970s by Newell and Simon, was the *problem space theory*, which takes the view that the mind is a limited information processor. Later variations on this drew on the earlier theory and attempted to reinterpret Gestalt theory in terms of information-processing theories. We will look briefly at each of these views.

Gestalt theory

Gestalt psychologists were answering the claim, made by behaviorists, that problem solving is a matter of reproducing known responses or trial and error. This explanation was considered by the Gestalt school to be insufficient to account for human problem-solving behavior. Instead, they claimed, problem solving is both *productive* and *reproductive*. Reproductive problem solving draws on previous experience as the behaviorists claimed, but productive problem solving involves insight and restructuring of the problem. Indeed, reproductive problem solving could be a hindrance to finding a solution, since a person may ‘fixate’ on the known aspects of the problem and so be unable to see novel interpretations that might lead to a solution.

Gestalt psychologists backed up their claims with experimental evidence. Kohler provided evidence of apparent insight being demonstrated by apes, which he observed joining sticks together in order to reach food outside their cages [202]. However, this was difficult to verify since the apes had once been wild and so could have been using previous knowledge.

Other experiments observed human problem-solving behavior. One well-known example of this is Maier’s *pendulum problem* [224]. The problem was this: the subjects were in a room with two pieces of string hanging from the ceiling. Also in the room were other objects including pliers, poles and extensions. The task set was to tie the pieces of string together. However, they were too far apart to catch hold of both at once. Although various solutions were proposed by subjects, few chose to use the weight of the pliers as a pendulum to ‘swing’ the strings together. However, when the experimenter brushed against the string, setting it in motion, this solution presented itself to subjects. Maier interpreted this as an example of productive restructuring. The movement of the string had given insight and allowed the subjects to see the problem in a new way. The experiment also illustrates fixation: subjects were initially unable to see beyond their view of the role or use of a pair of pliers.

Although Gestalt theory is attractive in terms of its description of human problem solving, it does not provide sufficient evidence or structure to support its theories. It does not explain when restructuring occurs or what insight is, for example. However, the move away from behaviorist theories was helpful in paving the way for the information-processing theory that was to follow.

Problem space theory

Newell and Simon proposed that problem solving centers on the problem space. The problem space comprises *problem states*, and problem solving involves generating these states using legal state transition operators. The problem has an initial state and a goal state and people use the operators to move from the former to the latter. Such problem spaces may be huge, and so *heuristics* are employed to select appropriate operators to reach the goal. One such heuristic is *means–ends analysis*. In means–ends analysis the initial state is compared with the goal state and an operator chosen to reduce the difference between the two. For example, imagine you are reorganizing your office and you want to move your desk from the north wall of the room to the window. Your initial state is that the desk is at the north wall. The goal state is that the desk is by the window. The main difference between these two is the location of your desk. You have a number of operators which you can apply to moving things: you can carry them or push them or drag them, etc. However, you know that to carry something it must be light and that your desk is heavy. You therefore have a new subgoal: to make the desk light. Your operators for this may involve removing drawers, and so on.

An important feature of Newell and Simon's model is that it operates within the constraints of the human processing system, and so searching the problem space is limited by the capacity of short-term memory, and the speed at which information can be retrieved. Within the problem space framework, experience allows us to solve problems more easily since we can structure the problem space appropriately and choose operators efficiently.

Newell and Simon's theory, and their *General Problem Solver* model which is based on it, have largely been applied to problem solving in well-defined domains, for example solving puzzles. These problems may be unfamiliar but the knowledge that is required to solve them is present in the statement of the problem and the expected solution is clear. In real-world problems finding the knowledge required to solve the problem may be part of the problem, or specifying the goal may be difficult. Problems such as these require significant domain knowledge: for example, to solve a programming problem you need knowledge of the language and the domain in which the program operates. In this instance specifying the goal clearly may be a significant part of solving the problem.

However, the problem space framework provides a clear theory of problem solving, which can be extended, as we shall see when we look at skill acquisition in the next section, to deal with knowledge-intensive problem solving. First we will look briefly at the use of analogy in problem solving.

Worked exercise *Identify the goals and operators involved in the problem 'delete the second paragraph of the document' on a word processor. Now use a word processor to delete a paragraph and note your actions, goals and subgoals. How well did they match your earlier description?*

Answer Assume you have a document open and you are at some arbitrary position within it. You also need to decide which operators are available and what their preconditions and results are. Based on an imaginary word processor we assume the following operators (you may wish to use your own WP package):

Operator	Precondition	Result
delete_paragraph	Cursor at start of paragraph	Paragraph deleted
move_to_paragraph	Cursor anywhere in document	Cursor moves to start of next paragraph (except where there is no next paragraph when no effect)
move_to_start	Cursor anywhere in document	Cursor at start of document

Goal: *delete second paragraph in document*
Looking at the operators an obvious one to resolve this goal is delete_paragraph which has the precondition 'cursor at start of paragraph'. We therefore have a new subgoal: *move_to_paragraph*. The precondition is 'cursor anywhere in document' (which we can meet) but we want the second paragraph so we must initially be in the first.

We set up a new subgoal, move_to_start, with precondition 'cursor anywhere in document' and result 'cursor at start of document'. We can then apply *move_to_paragraph* and finally *delete_paragraph*.

We assume some knowledge here (that the second paragraph is the paragraph after the first one).

Analogy in problem solving

A third element of problem solving is the use of analogy. Here we are interested in how people solve novel problems. One suggestion is that this is done by mapping knowledge relating to a similar known domain to the new problem – called *analogical mapping*. Similarities between the known domain and the new one are noted and operators from the known domain are transferred to the new one.

This process has been investigated using analogous stories. Gick and Holyoak [149] gave subjects the following problem:

> A doctor is treating a malignant tumor. In order to destroy it he needs to blast it with high-intensity rays. However, these will also destroy the healthy tissue surrounding the tumor. If he lessens the rays' intensity the tumor will remain. How does he destroy the tumor?

The solution to this problem is to fire low-intensity rays from different directions converging on the tumor. That way, the healthy tissue receives harmless low-intensity rays while the tumor receives the rays combined, making a high-intensity dose. The investigators found that only 10% of subjects reached this solution without help. However, this rose to 80% when they were given this analogous story and told that it may help them:

> A general is attacking a fortress. He can't send all his men in together as the roads are mined to explode if large numbers of men cross them. He therefore splits his men into small groups and sends them in on separate roads.

In spite of this, it seems that people often miss analogous information, unless it is semantically close to the problem domain. When subjects were not told to use the story, many failed to see the analogy. However, the number spotting the analogy rose when the story was made semantically close to the problem, for example a general using rays to destroy a castle.

The use of analogy is reminiscent of the Gestalt view of productive restructuring and insight. Old knowledge is used to solve a new problem.

1.4.3 Skill acquisition

All of the problem solving that we have considered so far has concentrated on handling unfamiliar problems. However, for much of the time, the problems that we face are not completely new. Instead, we gradually acquire skill in a particular domain area. But how is such skill acquired and what difference does it make to our problem-solving performance? We can gain insight into how skilled behavior works, and how skills are acquired, by considering the difference between novice and expert behavior in given domains.

Chess: of human and artificial intelligence

A few years ago, Deep Blue, a chess-playing computer, beat Gary Kasparov, the world's top Grand Master, in a full tournament. This was the long-awaited breakthrough for the artificial intelligence (AI) community, who have traditionally seen chess as the ultimate test of their art. However, despite the fact that computer chess programs can play at Grand Master level against human players, this does not mean they play in the same way. For each move played, Deep Blue investigated many millions of alternative moves and counter-moves. In contrast, a human chess player will only consider a few dozen. But, if the human player is good, these will usually be the right few dozen. The ability to spot patterns allows a human to address a problem with far less effort than a brute force approach. In chess, the number of moves is such that finally brute force, applied fast enough, has overcome human pattern-matching skill. In Go, which has far more possible moves, computer programs do not even reach a good club level of play. Many models of the mental processes have been heavily influenced by computation. It is worth remembering that although there are similarities, computer 'intelligence' is very different from that of humans.

A commonly studied domain is chess playing. It is particularly suitable since it lends itself easily to representation in terms of problem space theory. The initial state is the opening board position; the goal state is one player checkmating the other; operators to move states are legal moves of chess. It is therefore possible to examine skilled behavior within the context of the problem space theory of problem solving.

Studies of chess players by DeGroot, Chase and Simon, among others, produced some interesting observations [64, 65, 88, 89]. In all the experiments the behavior of chess masters was compared with less experienced chess players. The first observation was that players did not consider large numbers of moves in choosing their move, nor did they look ahead more than six moves (often far fewer). Masters considered no more alternatives than the less experienced, but they took less time to make a decision and produced better moves.

So what makes the difference between skilled and less skilled behavior in chess? It appears that chess masters remember board configurations and good moves associated with them. When given actual board positions to remember, masters are much better at reconstructing the board than the less experienced. However, when given random configurations (which were unfamiliar), the groups of players were equally bad at reconstructing the positions. It seems therefore that expert players 'chunk' the board configuration in order to hold it in short-term memory. Expert players use larger chunks than the less experienced and can therefore remember more detail.

This behavior is also seen among skilled computer programmers. They can also reconstruct programs more effectively than novices since they have the structures available to build appropriate chunks. They acquire plans representing code to solve particular problems. When that problem is encountered in a new domain or new program they will recall that particular plan and reuse it.

Another observed difference between skilled and less skilled problem solving is in the way that different problems are grouped. Novices tend to group problems according to superficial characteristics such as the objects or features common to both. Experts, on the other hand, demonstrate a deeper understanding of the problems and group them according to underlying conceptual similarities which may not be at all obvious from the problem descriptions.

Each of these differences stems from a better encoding of knowledge in the expert: information structures are fine tuned at a deep level to enable efficient and accurate retrieval. But how does this happen? How is skill such as this acquired? One model of skill acquisition is Anderson's *ACT** model [14]. ACT* identifies three basic levels of skill:

1. The learner uses general-purpose rules which interpret facts about a problem. This is slow and demanding on memory access.
2. The learner develops rules specific to the task.
3. The rules are tuned to speed up performance.

General mechanisms are provided to account for the transitions between these levels. For example, *proceduralization* is a mechanism to move from the first to the second. It removes the parts of the rule which demand memory access and replaces variables with specific values. *Generalization*, on the other hand, is a mechanism which moves from the second level to the third. It generalizes from the specific cases to general properties of those cases. Commonalities between rules are condensed to produce a general-purpose rule.

These are best illustrated by example. Imagine you are learning to cook. Initially you may have a general rule to tell you how long a dish needs to be in the oven, and a number of explicit representations of dishes in memory. You can instantiate the rule by retrieving information from memory.

```
IF cook[type, ingredients, time]
THEN
   cook for: time
cook[casserole, [chicken,carrots,potatoes], 2 hours]
cook[casserole, [beef,dumplings,carrots], 2 hours]
cook[cake, [flour,sugar,butter,eggs], 45 mins]
```

Gradually your knowledge becomes proceduralized and you have specific rules for each case:

```
IF type is casserole
AND ingredients are [chicken,carrots,potatoes]
THEN
   cook for: 2 hours
IF type is casserole
AND ingredients are [beef,dumplings,carrots]
THEN
   cook for: 2 hours
IF type is cake
AND ingredients are [flour,sugar,butter,eggs]
THEN
   cook for: 45 mins
```

Finally, you may generalize from these rules to produce general-purpose rules, which exploit their commonalities:

```
IF type is casserole
AND ingredients are ANYTHING
THEN
   cook for: 2 hours
```

The first stage uses knowledge extensively. The second stage relies upon known procedures. The third stage represents skilled behavior. Such behavior may in fact become automatic and as such be difficult to make explicit. For example, think of an activity at which you are skilled, perhaps driving a car or riding a bike. Try to describe to someone the exact procedure which you go through to do this. You will find this quite difficult. In fact experts tend to have to rehearse their actions mentally in order to identify exactly what they do. Such skilled behavior is efficient but may cause errors when the context of the activity changes.

1.4.4 Errors and mental models

Human capability for interpreting and manipulating information is quite impressive. However, we do make mistakes. Some are trivial, resulting in no more than temporary inconvenience or annoyance. Others may be more serious, requiring substantial effort to correct. Occasionally an error may have catastrophic effects, as we see when 'human error' results in a plane crash or nuclear plant leak.

Why do we make mistakes and can we avoid them? In order to answer the latter part of the question we must first look at what is going on when we make an error. There are several different types of error. As we saw in the last section some errors result from changes in the context of skilled behavior. If a pattern of behavior has become automatic and we change some aspect of it, the more familiar pattern may break through and cause an error. A familiar example of this is where we intend to stop at the shop on the way home from work but in fact drive past. Here, the activity of driving home is the more familiar and overrides the less familiar intention.

Other errors result from an incorrect understanding, or model, of a situation or system. People build their own theories to understand the causal behavior of systems. These have been termed *mental models*. They have a number of characteristics. Mental models are often partial: the person does not have a full understanding of the working of the whole system. They are unstable and are subject to change. They can be internally inconsistent, since the person may not have worked through the logical consequences of their beliefs. They are often unscientific and may be based on superstition rather than evidence. Often they are based on an incorrect interpretation of the evidence.

DESIGN FOCUS

Human error and false memories

In the second edition of this book, one of the authors added the following story:

> During the Second World War a new cockpit design was introduced for Spitfires. The pilots were trained and flew successfully during training, but would unaccountably bail out when engaged in dog fights. The new design had exchanged the positions of the gun trigger and ejector controls. In the heat of battle the old responses resurfaced and the pilots ejected. Human error, yes, but the designer's error, not the pilot's.

It is a good story, but after the book was published we got several emails saying 'Spitfires didn't have ejector seats'. It was Kai-Mikael Jää-Aro who was able to find what may have been the original to the story (and incidentally inform us what model of Spitfire was in our photo and who the pilot was!). He pointed us to and translated the story of Sierra 44, an S35E Draken reconnaissance aircraft.[1] The full story involves just about every perceptual and cognitive error imaginable, but the point that links to

1. Pej Kristoffersson, 1984. Sigurd 44 – Historien om hur man gör bort sig så att det märks by, *Flygrevyn* 2/1984, pp. 44–6.

the (false) Spitfire story is that in the Draken the red buttons for releasing the fuel 'drop' tanks and for the canopy release differed only in very small writing. In an emergency (burning fuel tanks) the pilot accidentally released the canopy and so ended up flying home cabriolet style.

There is a second story of human error here – the author's memory. When the book was written he could not recall where he had come across the story but was convinced it was to do with a Spitfire. It may be that he had been told the story by someone else who had got it mixed up, but it is as likely that he simply remembered the rough outline of the story and then 'reconstructed' the rest. In fact that is exactly how our memories work. Our brains do not bother to lay down every little detail, but when we 'remember' we rebuild what the incident 'must have been' using our world knowledge. This process is completely unconscious and can lead to what are known as *false memories*. This is particularly problematic in witness statements in criminal trials as early questioning by police or lawyers can unintentionally lead to witnesses being sure they have seen things that they have not. Numerous controlled psychological experiments have demonstrated this effect which furthermore is strongly influenced by biasing factors such as the race of supposed criminals.

To save his blushes we have not said here which author's failing memory was responsible for the Spitfire story, but you can read more on this story and also find who it was on the book website at: /e3/online/spitfire/

Courtesy of popperfoto.com

Assuming a person builds a mental model of the system being dealt with, errors may occur if the actual operation differs from the mental model. For example, on one occasion we were staying in a hotel in Germany, attending a conference. In the lobby of the hotel was a lift. Beside the lift door was a button. Our model of the system, based on previous experience of lifts, was that the button would call the lift. We pressed the button and the lobby light went out! In fact the button was a light switch and the lift button was on the inside rim of the lift, hidden from view.

Although both the light switch and the lift button were inconsistent with our mental models of these controls, we would probably have managed if they had been encountered separately. If there had been no button beside the lift we would have looked more closely and found the one on the inner rim. But since the light switch reflected our model of a lift button we looked no further. During our stay we observed many more new guests making the same error.

This illustrates the importance of a correct mental model and the dangers of ignoring conventions. There are certain conventions that we use to interpret the world and ideally designs should support these. If these are to be violated, explicit support must be given to enable us to form a correct mental model. A label on the button saying 'light switch' would have been sufficient.

1.5 EMOTION

So far in this chapter we have concentrated on human perceptual and cognitive abilities. But human experience is far more complex than this. Our emotional response to situations affects how we perform. For example, positive emotions enable us to think more creatively, to solve complex problems, whereas negative emotion pushes us into narrow, focussed thinking. A problem that may be easy to solve when we are relaxed, will become difficult if we are frustrated or afraid.

Psychologists have studied emotional response for decades and there are many theories as to what is happening when we feel an emotion and why such a response occurs. More than a century ago, William James proposed what has become known as the James–Lange theory (Lange was a contemporary of James whose theories were similar): that emotion was the interpretation of a physiological response, rather than the other way around. So while we may feel that we respond *to* an emotion, James contended that we respond physiologically to a stimulus and interpret that as emotion:

> Common sense says, we lose our fortune, are sorry and weep; we meet a bear, are frightened and run; we are insulted by a rival, are angry and strike. The hypothesis here . . . is that we feel sorry because we cry, angry because we strike, afraid because we tremble.
>
> (W. James, *Principles of Psychology*, page 449. Henry Holt, New York, 1890.)

Others, however, disagree. Cannon [54a], for example, argued that our physiological processes are in fact too slow to account for our emotional reactions, and that the physiological responses for some emotional states are too similar (e.g. anger and fear), yet they can be easily distinguished. Experience in studies with the use of drugs that stimulate broadly the same physiological responses as anger or fear seems to support this: participants reported physical symptoms but not the emotion, which suggests that emotional response is more than a recognition of physiological changes.

Schachter and Singer [312a] proposed a third interpretation: that emotion results from a person evaluating physical responses in the light of the whole situation. So whereas the same physiological response can result from a range of different situations, the emotion that is felt is based on a cognitive evaluation of the circumstance and will depend on what the person attributes this to. So the same physiological response of a pounding heart will be interpreted as excitement if we are in a competition and fear if we find ourselves under attack.

Whatever the exact process, what is clear is that emotion involves both physical and cognitive events. Our body responds biologically to an external stimulus and we interpret that in some way as a particular emotion. That biological response – known as *affect* – changes the way we deal with different situations, and this has an impact on the way we interact with computer systems. As Donald Norman says:

> Negative affect can make it harder to do even easy tasks; positive affect can make it easier to do difficult tasks.
>
> (D. A. Norman, Emotion and design: attractive things work better. *Interactions Magazine*, ix(4): 36–42, 2002.)

So what are the implications of this for design? It suggests that in situations of stress, people will be less able to cope with complex problem solving or managing difficult interfaces, whereas if people are relaxed they will be more forgiving of limitations in the design. This does not give us an excuse to design bad interfaces but does suggest that if we build interfaces that promote positive responses – for example by using aesthetics or reward – then they are likely be more successful.

1.6 INDIVIDUAL DIFFERENCES

In this chapter we have been discussing humans in general. We have made the assumption that everyone has similar capabilities and limitations and that we can therefore make generalizations. To an extent this is true: the psychological principles and properties that we have discussed apply to the majority of people. Notwithstanding this, we should remember that, although we share processes in common, humans, and therefore users, are not all the same. We should be aware of individual differences so that we can account for them as far as possible within our designs. These differences may be long term, such as sex, physical capabilities and intellectual capabilities. Others are shorter term and include the effect of stress or fatigue on the user. Still others change through time, such as age.

These differences should be taken into account in our designs. It is useful to consider, for any design decision, if there are likely to be users within the target group who will be adversely affected by our decision. At the extremes a decision may exclude a section of the user population. For example, the current emphasis on visual interfaces excludes those who are visually impaired, unless the design also makes use of the other sensory channels. On a more mundane level, designs should allow for users who are under pressure, feeling ill or distracted by other concerns: they should not push users to their perceptual or cognitive limits.

We will consider the issues of universal accessibility in more detail in Chapter 10.

1.7 PSYCHOLOGY AND THE DESIGN OF INTERACTIVE SYSTEMS

So far we have looked briefly at the way in which humans receive, process and store information, solve problems and acquire skill. But how can we apply what we have learned to designing interactive systems? Sometimes, straightforward conclusions can be drawn. For example, we can deduce that recognition is easier than recall and allow users to select commands from a set (such as a menu) rather than input them directly. However, in the majority of cases, application is not so obvious or simple. In fact, it may be dangerous, leading us to make generalizations which are not valid. In order to apply a psychological principle or result properly in design, we need to understand its context, both in terms of where it fits in the wider field of psychology and in terms of the details of the actual experiments, the measures used and the subjects involved, for example. This may appear daunting, particularly to the novice designer who wants to acknowledge the relevance of cognitive psychology but does not have the background to derive appropriate conclusions. Fortunately, principles and results from research in psychology have been distilled into guidelines for design, models to support design and techniques for evaluating design. Parts 2 and 3 of this book include discussion of a range of guidelines, models and techniques, based on cognitive psychology, which can be used to support the design process.

1.7.1 Guidelines

Throughout this chapter we have discussed the strengths and weaknesses of human cognitive and perceptual processes but, for the most part, we have avoided attempting to apply these directly to design. This is because such an attempt could only be partial and simplistic, and may give the impression that this is all psychology has to offer.

However, general design principles and guidelines can be and have been derived from the theories we have discussed. Some of these are relatively straightforward: for instance, recall is assisted by the provision of retrieval cues so interfaces should incorporate recognizable cues wherever possible. Others are more complex and context dependent. In Chapter 7 we discuss principles and guidelines further, many of which are derived from psychological theory. The interested reader is also referred to Gardiner and Christie [140] which illustrates how guidelines can be derived from psychological theory.

1.7.2 Models to support design

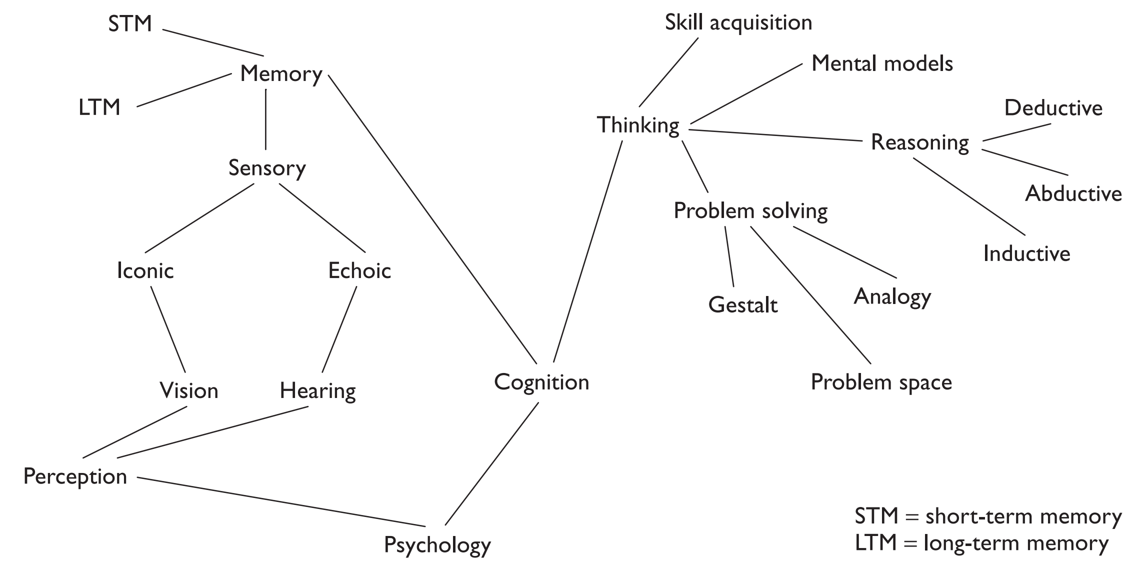
As well as guidelines and principles, psychological theory has led to the development of analytic and predictive models of user behavior. Some of these include a specific model of human problem solving, others of physical activity, and others attempt a more comprehensive view of cognition. Some predict how a typical computer user would behave in a given situation, others analyze why particular user behavior occurred. All are based on cognitive theory. We discuss these models in detail in Chapter 12.

1.7.3 Techniques for evaluation

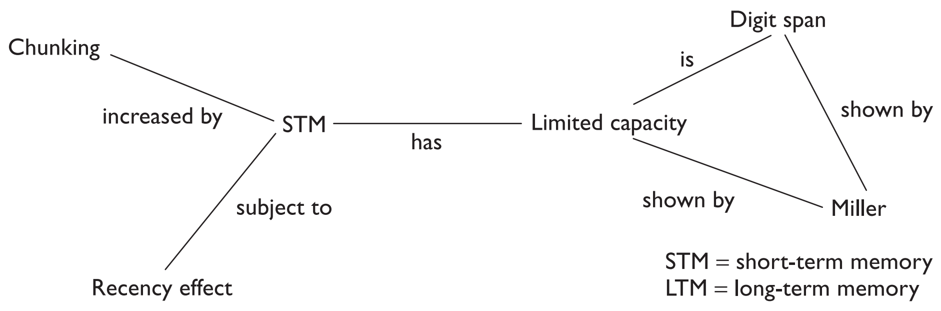
In addition to providing us with a wealth of theoretical understanding of the human user, psychology also provides a range of empirical techniques which we can employ to evaluate our designs and our systems. In order to use these effectively we need to understand the scope and benefits of each method. Chapter 9 provides an overview of these techniques and an indication of the circumstances under which each should be used.

Worked exercise *Produce a semantic network of the main information in this chapter.*

Answer This network is potentially huge so it is probably unnecessary to devise the whole thing! Be selective. One helpful way to tackle the exercise is to approach it in both a top-down and a bottom-up manner. Top-down will give you a general overview of topics and how they relate; bottom-up can fill in the details of a particular field. These can then be



Top-down view



Bottom-up view

'glued' together to build up the whole picture. You may be able to tackle this problem in a group, each taking one part of it. We will not provide the full network here but will give examples of the level of detail anticipated for the overview and the detailed versions. In the overview we have not included labels on the arcs for clarity.

1.8 SUMMARY

In this chapter we have considered the human as an information processor, receiving inputs from the world, storing, manipulating and using information, and reacting to the information received. Information is received through the senses, particularly, in the case of computer use, through sight, hearing and touch. It is stored in memory, either temporarily in sensory or working memory, or permanently in long-term memory. It can then be used in reasoning and problem solving. Recurrent familiar situations allow people to acquire skills in a particular domain, as their information structures become better defined. However, this can also lead to error, if the context changes.

Human perception and cognition are complex and sophisticated but they are not without their limitations. We have considered some of these limitations in this chapter. An understanding of the capabilities and limitations of the human as information processor can help us to design interactive systems which support the former and compensate for the latter. The principles, guidelines and models which can be derived from cognitive psychology and the techniques which it provides are invaluable tools for the designer of interactive systems.

EXERCISES

1.1 Devise experiments to test the properties of (i) short-term memory, (ii) long-term memory, using the experiments described in this chapter to help you. Try out your experiments on your friends. Are your results consistent with the properties described in this chapter?

1.2 Observe skilled and novice operators in a familiar domain, for example touch and 'hunt-and-peck' typists, expert and novice game players, or expert and novice users of a computer application. What differences can you discern between their behaviors?

1.3 From what you have learned about cognitive psychology devise appropriate guidelines for use by interface designers. You may find it helpful to group these under key headings, for example visual perception, memory, problem solving, etc., although some may overlap such groupings.

1.4 What are *mental models*, and why are they important in interface design?

1.5 What can a system designer do to minimize the memory load of the user?

1.6 Human short-term memory has a limited span. This is a series of experiments to determine what that span is. (You will need some other people to take part in these experiments with you – they do not need to be studying the course – try it with a group of friends.)

(a) *Kim's game*
Divide into groups. Each group gathers together an assortment of objects – pens, pencils, paper-clips, books, sticky notes, etc. The stranger the object, the better! You need a large number of them – at least 12 to 15. Place them in some compact arrangement on a table, so that all items are visible. Then, swap with another group for 30 seconds only and look at their pile. Return to your table, and on your own try to write down all the items in the other group's pile.

Compare your list with what they actually have in their pile. Compare the number of things you remembered with how the rest of your group did. Now think introspectively: what helped you remember certain things? Did you recognize things in their pile that you had in yours? Did that help? Do not pack the things away just yet.

Calculate the average score for your group. Compare that with the averages from the other group(s).

Questions: What conclusions can you draw from this experiment? What does this indicate about the capacity of short-term memory? What does it indicate that helps improve the capacity of short-term memory?

(b) *'I went to market . . .'*
In your group, one person starts off with 'I went to market and I bought a fish' (or some other produce, or whatever!). The next person continues 'I went to market and I bought a fish and I bought a bread roll as well'. The process continues, with each person adding some item to the list each time. Keep going around the group until you cannot remember the list accurately. Make a note of the first time someone gets it wrong, and then record the number of items that you can successfully remember. Some of you will find it hard to remember more than a few, others will fare much better. Do this a few more times with different lists, and then calculate your average score, and your group's average score.

Questions: What does this tell you about short-term memory? What do you do that helps you remember? What do you estimate is the typical capacity of human short-term memory? Is this a good test for short-term memory?

(c) *Improving your memory*
Try experiment 1.6(a) again, using the techniques on page 39.

Has your recall ability improved? Has your group's average improved? What does this show you about memory?

1.7 Locate one source (through the library or the web) that reports on empirical evidence on human limitations. Provide a full reference to the source. In one paragraph, summarize what the result of the research states in terms of a physical human limitation.

In a separate paragraph, write your thoughts on how you think this evidence on human capabilities impacts interactive system design.

RECOMMENDED READING

E. B. Goldstein, *Sensation and Perception*, 6th edition, Wadsworth, 2001.
A textbook covering human senses and perception in detail. Easy to read with many home experiments to illustrate the points made.

A. Baddeley, *Human Memory: Theory and Practice*, revised edition, Allyn & Bacon, 1997.
The latest and most complete of Baddeley's texts on memory. Provides up-to-date discussion on the different views of memory structure as well as a detailed survey of experimental work on memory.

M. W. Eysenck and M. T. Keane, *Cognitive Psychology: A Student's Handbook*, 4th edition, Psychology Press, 2000.
A comprehensive and readable textbook giving more detail on cognitive psychology, including memory, problem solving and skill acquisition.

S. K. Card, T. P. Moran and A. Newell, *The Psychology of Human–Computer Interaction*, Lawrence Erlbaum Associates, 1983.
A classic text looking at the human as an information processor in interaction with the computer. Develops and describes the Model Human Processor in detail.

A. Newell and H. Simon, *Human Problem Solving*, Prentice Hall, 1972.
Describes the problem space view of problem solving in more detail.

M. M. Gardiner and B. Christie, editors, *Applying Cognitive Psychology to User-Interface Design*, John Wiley, 1987.
A collection of essays on the implications of different aspects of cognitive psychology to interface design. Includes memory, thinking, language and skill acquisition. Provides detailed guidelines for applying psychological principles in design practice.

A. Monk, editor, *Fundamentals of Human Computer Interaction*, Academic Press, 1985.
A good collection of articles giving brief coverage of aspects of human psychology including perception, memory, thinking and reading. Also contains articles on experimental design which provide useful introductions.

ACT-R site. Website of resources and examples of the use of the cognitive architecture ACT-R, which is the latest development of Anderson's ACT model, http://act-r.psy.cmu.edu/

2 The computer

OVERVIEW

A computer system comprises various elements, each of which affects the user of the system.

- Input devices for interactive use, allowing text entry, drawing and selection from the screen:
  - text entry: traditional keyboard, phone text entry, speech and handwriting
  - pointing: principally the mouse, but also touchpad, stylus and others
  - 3D interaction devices.
- Output display devices for interactive use:
  - different types of screen mostly using some form of bitmap display
  - large displays and situated displays for shared and public use
  - digital paper may be usable in the near future.
- Virtual reality systems and 3D visualization which have special interaction and display devices.
- Various devices in the physical world:
  - physical controls and dedicated displays
  - sound, smell and haptic feedback
  - sensors for nearly everything including movement, temperature, bio-signs.
- Paper output and input: the paperless office and the less-paper office:
  - different types of printers and their characteristics, character styles and fonts
  - scanners and optical character recognition.
- Memory:
  - short-term memory: RAM
  - long-term memory: magnetic and optical disks
  - capacity limitations related to document and video storage
  - access methods as they limit or help the user.
- Processing:
  - the effects when systems run too slow or too fast, the myth of the infinitely fast machine
  - limitations on processing speed
  - networks and their impact on system performance.

2.1 INTRODUCTION

In order to understand how humans interact with computers, we need to have an understanding of both parties in the interaction. The previous chapter explored aspects of human capabilities and behavior of which we need to be aware in the context of human–computer interaction; this chapter considers the computer and associated input–output devices and investigates how the technology influences the nature of the interaction and style of the interface.

We will concentrate principally on the traditional computer but we will also look at devices that take us beyond the closed world of keyboard, mouse and screen. As well as giving us lessons about more traditional systems, these are increasingly becoming important application areas in HCI.

Exercise: how many computers?

In a group or class do a quick survey:

- How many computers do you have in your home?
- How many computers do you normally carry with you in your pockets or bags?

Collate the answers and see who the techno-freaks are!

Discuss your answers.

After doing this look at /e3/online/how-many-computers/

When we interact with computers, what are we trying to achieve? Consider what happens when we interact with each other – we are either passing information to other people, or receiving information from them. Often, the information we receive is in response to the information that we have recently imparted to them, and we may then respond to that. Interaction is therefore a process of information transfer. Relating this to the electronic computer, the same principles hold: interaction is a process of information transfer, from the user to the computer and from the computer to the user.

The first part of this chapter concentrates on the transference of information from the user to the computer and back. We begin by considering a current typical computer interface and the devices it employs, largely variants of keyboard for text entry (Section 2.2), mouse for positioning (Section 2.3) and screen for displaying output (Section 2.4).

Then we move on to consider devices that go beyond the keyboard, mouse and screen: entering deeper into the electronic world with virtual reality and 3D interaction

(Section 2.5) and outside the electronic world looking at more physical interactions (Section 2.6).

In addition to direct input and output, information is passed to and fro via paper documents. This is dealt with in Section 2.7, which describes printers and scanners. Although not requiring the same degree of user interaction as a mouse or keyboard, these are an important means of input and output for many current applications.

We then consider the computer itself, its processor and memory devices and the networks that link them together. We note how the technology drives and empowers the interface. The details of computer processing should largely be irrelevant to the end-user, but the interface designer needs to be aware of the limitations of storage capacity and computational power; it is no good designing on paper a marvellous new interface, only to find it needs a Cray to run. Software designers often have high-end machines on which to develop applications, and it is easy to forget what a more typical configuration feels like.

Before looking at these devices and technology in detail we'll take a quick bird's-eye view of the way computer systems are changing.

2.1.1 A typical computer system

Consider a typical computer setup as shown in Figure 2.1. There is the computer 'box' itself, a keyboard, a mouse and a color screen. The screen layout is shown alongside it. If we examine the interface, we can see how its various characteristics are related to the devices used. The details of the interface itself, its underlying principles and design, are discussed in more depth in Chapter 3. As we shall see there are variants on these basic devices. Some of this variation is driven by different hardware configurations: desktop use, laptop computers, PDAs (personal digital assistants). Partly the diversity of devices reflects the fact that there are many different types of

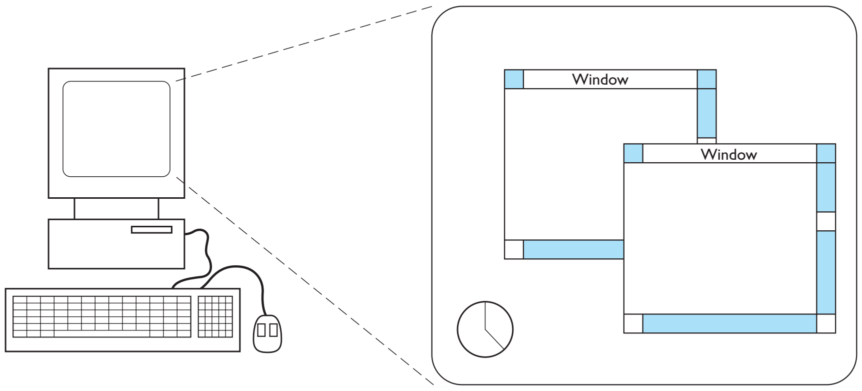


Figure 2.1 A typical computer system

data that may have to be entered into and obtained from a system, and there are also many different types of user, each with their own unique requirements.

2.1.2 Levels of interaction – batch processing

In the early days of computing, information was entered into the computer in a large mass – batch data entry. There was minimal interaction with the machine: the user would simply dump a pile of punched cards onto a reader, press the start button, and then return a few hours later. This still continues today although now with pre-prepared electronic files or possibly machine-read forms. It is clearly the most appropriate mode for certain kinds of application, for example printing pay checks or entering the results from a questionnaire.

With batch processing the interactions take place over hours or days. In contrast the typical desktop computer system has interactions taking seconds or fractions of a second (or with slow web pages sometimes minutes!). The field of Human–Computer Interaction largely grew due to this change in interactive pace. It is easy to assume that faster means better, but some of the paper-based technology discussed in Section 2.7 suggests that sometimes slower paced interaction may be better.

2.1.3 Richer interaction – everywhere, everywhen

Computers are coming out of the box! Information appliances are putting internet access or dedicated systems onto the fridge, microwave and washing machine: to automate shopping, give you email in your kitchen or simply call for maintenance when needed. We carry with us WAP phones and smartcards, have security systems that monitor us and web cams that show our homes to the world. Is Figure 2.1 really the typical computer system or is it really more like Figure 2.2?

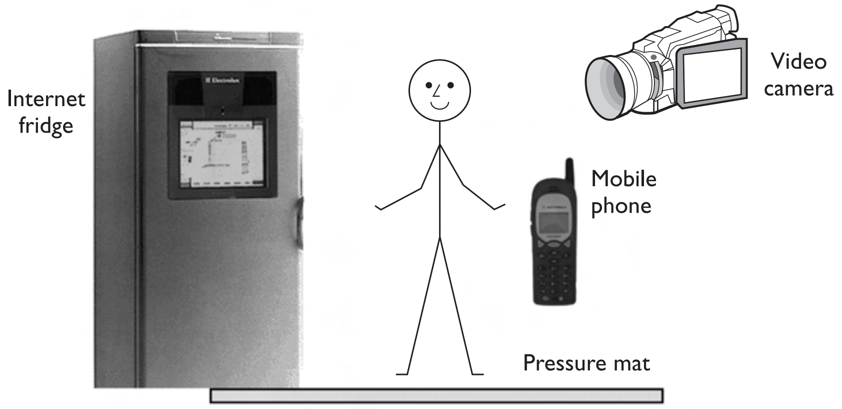


Figure 2.2 A typical computer system? Photo courtesy Electrolux

2.2 TEXT ENTRY DEVICES

Whether writing a book like this, producing an office memo, sending a thank you letter after your birthday, or simply sending an email to a friend, entering text is one of our main activities when using the computer. The most obvious means of text entry is the plain keyboard, but there are several variations on this: different keyboard layouts, 'chord' keyboards that use combinations of fingers to enter letters, and phone key pads. Handwriting and speech recognition offer more radical alternatives.

2.2.1 The alphanumeric keyboard

The keyboard is still one of the most common input devices in use today. It is used for entering textual data and commands. The vast majority of keyboards have a standardized layout, and are known by the first six letters of the top row of alphabetical keys, QWERTY. There are alternative designs which have some advantages over the QWERTY layout, but these have not been able to overcome the vast technological inertia of the QWERTY keyboard. These alternatives are of two forms: 26 key layouts and chord keyboards. A 26 key layout rearranges the order of the alphabetic keys, putting the most commonly used letters under the strongest fingers, or adopting simpler practices. In addition to QWERTY, we will discuss two 26 key layouts, alphabetic and DVORAK, and chord keyboards.

The QWERTY keyboard

The layout of the digits and letters on a QWERTY keyboard is fixed (see Figure 2.3), but non-alphanumeric keys vary between keyboards. For example, there is a difference between key assignments on British and American keyboards (in particular, above the 3 on the UK keyboard is the pound sign £, whilst on the US keyboard there is a dollar sign $). The standard layout is also subject to variation in the placement of brackets, backslashes and suchlike. In addition different national keyboards include accented letters and the traditional French layout places the main letters in different locations – the top line starts AZERTY.

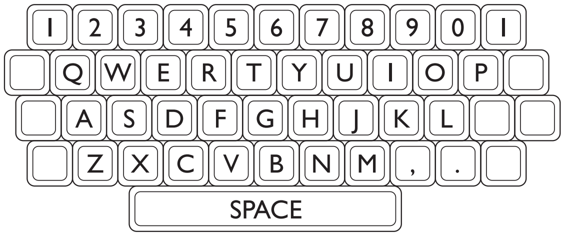


Figure 2.3 The standard QWERTY keyboard

The QWERTY arrangement of keys is not optimal for typing, however. The reason for the layout of the keyboard in this fashion can be traced back to the days of mechanical typewriters. Hitting a key caused an arm to shoot towards the carriage, imprinting the letter on the head on the ribbon and hence onto the paper. If two arms flew towards the paper in quick succession from nearly the same angle, they would often jam – the solution to this was to set out the keys so that common combinations of consecutive letters were placed at different ends of the keyboard, which meant that the arms would usually move from alternate sides. One appealing story relating to the key layout is that it was also important for a salesman to be able to type the word ‘typewriter’ quickly in order to impress potential customers: the letters are all on the top row!

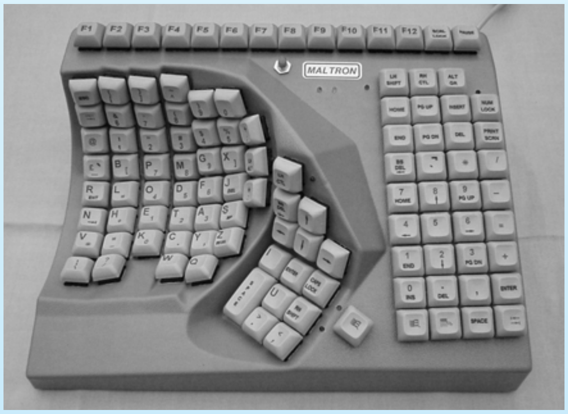
The electric typewriter and now the computer keyboard are not subject to the original mechanical constraints, but the QWERTY keyboard remains the dominant layout. The reason for this is social – the vast base of trained typists would be reluctant to relearn their craft, whilst the management is not prepared to accept an initial lowering of performance whilst the new skills are gained. There is also a large investment in current keyboards, which would all have to be either replaced at great cost, or phased out, with the subsequent requirement for people to be proficient on both keyboards. As whole populations have become keyboard users this technological inertia has probably become impossible to change.

How keyboards work

Current keyboards work by a keypress closing a connection, causing a character code to be sent to the computer. The connection is usually via a lead, but wireless systems also exist. One aspect of keyboards that is important to users is the ‘feel’ of the keys. Some keyboards require a very hard press to operate the key, much like a manual typewriter, whilst others are featherlight. The distance that the keys travel also affects the tactile nature of the keyboard. The keyboards that are currently used on most notebook computers are ‘half-travel’ keyboards, where the keys travel only a small distance before activating their connection; such a keyboard can feel dead to begin with, but such qualitative judgments often change as people become more used to using it. By making the actual keys thinner, and allowing them a much reduced travel, a lot of vertical space can be saved on the keyboard, thereby making the machine slimmer than would otherwise be possible.

Some keyboards are even made of touch-sensitive buttons, which require a light touch and practically no travel; they often appear as a sheet of plastic with the buttons printed on them. Such keyboards are often found on shop tills, though the keys are not QWERTY, but specific to the task. Being fully sealed, they have the advantage of being easily cleaned and resistant to dirty environments, but have little feel, and are not popular with trained touch-typists. Feedback is important even at this level of human–computer interaction! With the recent increase of repetitive strain injury (RSI) to users’ fingers, and the increased responsibilities of employers in these circumstances, it may be that such designs will enjoy a resurgence in the near future. RSI in fingers is caused by the tendons that control the movement of the fingers becoming inflamed owing to overuse and making repeated unnatural movements.

There are a variety of specially shaped keyboards to relieve the strain of typing or to allow people to type with some injury (e.g. RSI) or disability. These may slope the keys towards the hands to improve the ergonomic position, be designed for single-handed use, or for no hands at all. Some use bespoke key layouts to reduce strain of finger movements. The keyboard illustrated is produced by PCD Maltron Ltd. for left-handed use. See www.maltron.com/



Source: www.maltron.com, reproduced courtesy of PCD Maltron Ltd.

Ease of learning – alphabetic keyboard

One of the most obvious layouts to be produced is the alphabetic keyboard, in which the letters are arranged alphabetically across the keyboard. It might be expected that such a layout would make it quicker for untrained typists to use, but this is not the case. Studies have shown that this keyboard is not faster for properly trained typists, as we may expect, since there is no inherent advantage to this layout. And even for novice or occasional users, the alphabetic layout appears to make very little difference to the speed of typing. These keyboards are used in some pocket electronic personal organizers, perhaps because the layout looks simpler to use than the QWERTY one. Also, it dissuades people from attempting to use their touch-typing skills on a very small keyboard and hence avoids criticisms of difficulty of use!

Ergonomics of use – DVORAK keyboard and split designs

The DVORAK keyboard uses a similar layout of keys to the QWERTY system, but assigns the letters to different keys. Based upon an analysis of typing, the keyboard is designed to help people reach faster typing speeds. It is biased towards right-handed people, in that 56% of keystrokes are made with the right hand. The layout of the keys also attempts to ensure that the majority of keystrokes alternate between hands, thereby increasing the potential speed. By keeping the most commonly used keys on the home, or middle, row, 70% of keystrokes are made without the typist having to stretch far, thereby reducing fatigue and increasing keying speed. The layout also aims to minimize the number of keystrokes made with the weak fingers. Many of these requirements are in conflict, and the DVORAK keyboard represents one possible solution. Experiments have shown that there is a speed improvement of between 10 and 15%, coupled with a reduction in user fatigue due to the increased ergonomic layout of the keyboard [230].

Other aspects of keyboard design have been altered apart from the layout of the keys. A number of more ergonomic designs have appeared, in which the basic tilted planar base of the keyboard is altered. Moderate designs curve the plane of the keyboard, making it concave, whilst more extreme ones split the keys into those for the left and right hand and curve both halves separately. Often in these the keys are also moved to bring them all within easy reach, to minimize movement between keys. Such designs are supposed to aid comfort and reduce RSI by minimizing effort, but have had practically no impact on the majority of systems sold.

2.2.2 Chord keyboards

Chord keyboards are significantly different from normal alphanumeric keyboards. Only a few keys, four or five, are used (see Figure 2.4) and letters are produced by pressing one or more of the keys at once. For example, in the *Microwriter*, the pattern of multiple keypresses is chosen to reflect the actual letter shape.

Such keyboards have a number of advantages. They are extremely compact: simply reducing the size of a conventional keyboard makes the keys too small and close together, with a correspondingly large increase in the difficulty of using it. The

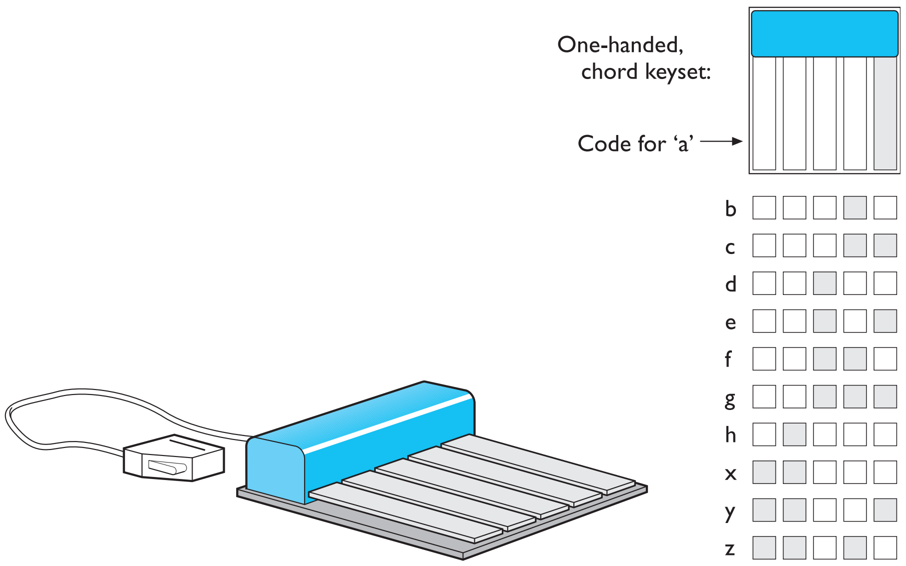


Figure 2.4 A very early chord keyboard (left) and its lettercodes (right)

learning time for the keyboard is supposed to be fairly short – of the order of a few hours – but social resistance is still high. Moreover, they are capable of fast typing speeds in the hands (or rather hand!) of a competent user. Chord keyboards can also be used where only one-handed operation is possible, in cramped and confined conditions.

Lack of familiarity means that these are unlikely ever to be a mainstream form of text entry, but they do have applications in niche areas. In particular, courtroom stenographers use a special form of two-handed chord keyboard and associated shorthand to enter text at full spoken speed. Also it may be that the compact size and one-handed operation will find a place in the growing wearables market.

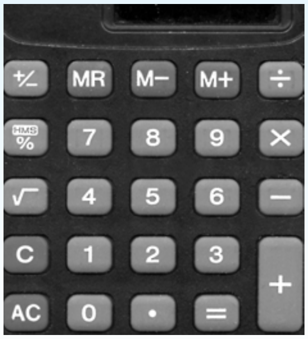
DESIGN FOCUS

Numeric keypads

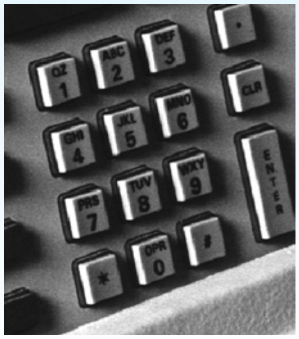
Alphanumeric keyboards (as the name suggests) include numbers as well as letters. In the QWERTY layout these are in a line across the top of the keyboard, but in most larger keyboards there is also a separate number pad to allow faster entry of digits. Number keypads occur in other contexts too, including calculators, telephones and ATM cash dispensers. Many people are unaware that there are two different layouts for numeric keypads: the calculator style that has '123' on the bottom and the telephone style that has '123' at the top.

It is a demonstration of the amazing adaptability of humans that we move between these two styles with such ease. However, if you need to include a numeric keypad in a device you must consider which is most appropriate for your potential users. For example, computer keyboards use calculator-style layout, as they are primarily used for entering numbers for calculations.

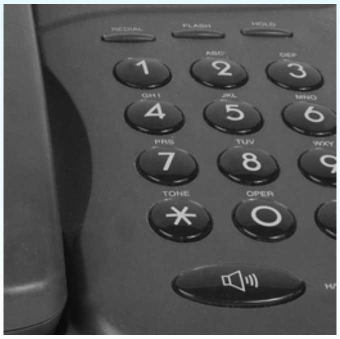
One of the authors was caught out by this once when he forgot the PIN number of his cash card. He half remembered the digits, but also his fingers knew where to type, so he 'practiced' on his calculator. Unfortunately ATMs use telephone-style layout!



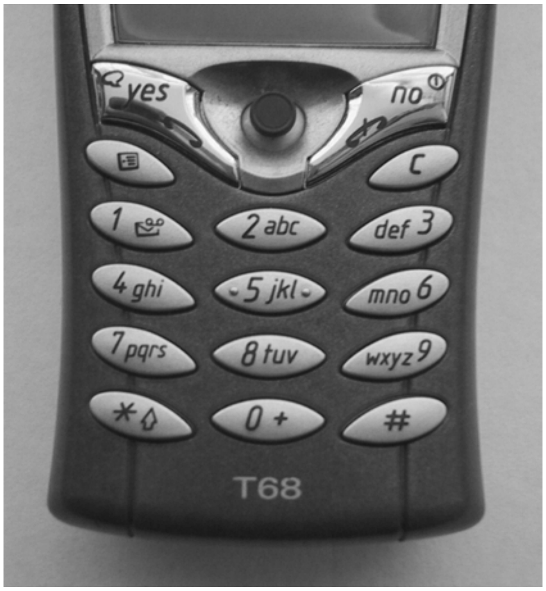
calculator



ATM



phone



Typical key mapping:
1 – space, comma, etc. (varies)
2 – a b c
3 – d e f
4 – g h i
5 – j k l
6 – m n o
7 – p q r s
8 – t u v
9 – w x y z
0 – +, &, etc.

Figure 2.5 Mobile phone keypad. Source: Photograph by Alan Dix (Ericsson phone)

2.2.3 Phone pad and T9 entry

With mobile phones being used for SMS text messaging (see Chapter 19) and WAP (see Chapter 21), the phone keypad has become an important form of text input. Unfortunately a phone only has digits 0–9, not a full alphanumeric keyboard.

To overcome this for text input the numeric keys are usually pressed several times – Figure 2.5 shows a typical mapping of digits to letters. For example, the 3 key has ‘def’ on it. If you press the key once you get a ‘d’, if you press 3 twice you get an ‘e’, if you press it three times you get an ‘f’. The main number-to-letter mapping is standard, but punctuation and accented letters differ between phones. Also there needs to be a way for the phone to distinguish, say, the ‘dd’ from ‘e’. On some phones you need to pause for a short period between successive letters using the same key, for others you press an additional key (e.g. ‘#’).

Most phones have at least two *modes* for the numeric buttons: one where the keys mean the digits (for example when entering a phone number) and one where they mean letters (for example when typing an SMS message). Some have additional modes to make entering accented characters easier. Also a special mode or setting is needed for capital letters although many phones use rules to reduce this, for example automatically capitalizing the initial letter in a message and letters following full stops, question marks and exclamation marks.

This is all very laborious and, as we will see in Chapter 19, experienced mobile phone users make use of a highly developed shorthand to reduce the number of keystrokes. If you watch a teenager or other experienced txt-er, you will see they often develop great typing speed holding the phone in one hand and using only their thumb. As these skills spread through society it may be that future devices use this as a means of small format text input. For those who never develop this physical dexterity some phones have tiny plug-in keyboards, or come with fold-out keyboards.

Another technical solution to the problem is the T9 algorithm. This uses a large dictionary to disambiguate words by simply typing the relevant letters once. For example, '3926753' becomes 'example' as there is only one word with letters that match (alternatives like 'ewbosld' that also match are not real words). Where there are ambiguities such as '26', which could be an 'am' or an 'an', the phone gives a series of options to choose from.

2.2.4 Handwriting recognition

Handwriting is a common and familiar activity, and is therefore attractive as a method of text entry. If we were able to write as we would when we use paper, but with the computer taking this form of input and converting it to text, we can see that it is an intuitive and simple way of interacting with the computer. However, there are a number of disadvantages with handwriting recognition. Current technology is still fairly inaccurate and so makes a significant number of mistakes in recognizing letters, though it has improved rapidly. Moreover, individual differences in handwriting are enormous, and make the recognition process even more difficult. The most significant information in handwriting is not in the letter shape itself but in the stroke information – the way in which the letter is drawn. This means that devices which support handwriting recognition must capture the stroke information, not just the final character shape. Because of this, online recognition is far easier than reading handwritten text on paper. Further complications arise because letters within words are shaped and often drawn very differently depending on the actual word; the context can help determine the letter's identity, but is often unable to provide enough information. Handwriting recognition is covered in more detail later in the book, in Chapter 10. More serious in many ways is the limitation on speed; it is difficult to write at more than 25 words a minute, which is no more than half the speed of a decent typist.

The different nature of handwriting means that we may find it more useful in situations where a keyboard-based approach would have its own problems. Such situations will invariably result in completely new systems being designed around the handwriting recognizer as the predominant mode of textual input, and these may bear very little resemblance to the typical system. Pen-based systems that use handwriting recognition are actively marketed in the mobile computing market, especially for smaller pocket organizers. Such machines are typically used for taking notes and jotting down and sketching ideas, as well as acting as a diary, address book and organizer. Using handwriting recognition has many advantages over using a keyboard. A pen-based system can be small and yet still accurate and easy to use, whereas small keys become very tiring, or even impossible, to use accurately. Also the pen-based approach does not have to be altered when we move from jotting down text to sketching diagrams; pen-based input is highly appropriate for this also.

Some organizer designs have dispensed with a keyboard completely. With such systems one must consider all sorts of other ways to interact with the system that are not character based. For example, we may decide to use *gesture recognition*, rather than commands, to tell the system what to do, for example drawing a line through a word in order to delete it. The important point is that a different input device that was initially considered simply as an alternative to the keyboard opens up a whole host of alternative interface designs and different possibilities for interaction.

Signature authentication

Handwriting recognition is difficult principally because of the great differences between different people's handwriting. These differences can be used to advantage in *signature authentication* where the purpose is to identify the user rather than read the signature. Again this is far easier when we have stroke information as people tend to produce signatures which *look* slightly different from one another in detail, but are formed in a similar fashion. Furthermore, a forger who has a copy of a person's signature may be able to copy the appearance of the signature, but will not be able to reproduce the pattern of strokes.

2.2.5 Speech recognition

Speech recognition is a promising area of text entry, but it has been promising for a number of years and is still only used in very limited situations. There is a natural enthusiasm for being able to talk to the machine and have it respond to commands, since this form of interaction is one with which we are very familiar. Successful recognition rates of over 97% have been reported, but since this represents one letter in error in approximately every 30, or one spelling mistake every six or so words, this is stoll unacceptable (*sic*)! Note also that this performance is usually quoted only for a restricted vocabulary of command words. Trying to extend such systems to the level of understanding natural language, with its inherent vagueness, imprecision and pauses, opens up many more problems that have not been satisfactorily solved even for keyboard-entered natural language. Moreover, since every person speaks differently, the system has to be trained and tuned to each new speaker, or its performance decreases. Strong accents, a cold or emotion can also cause recognition problems, as can background noise. This leads us on to the question of practicality within an office environment: not only may the background level of noise cause errors, but if everyone in an open-plan office were to talk to their machine, the level of noise would dramatically increase, with associated difficulties. Confidentiality would also be harder to maintain.

Despite its problems, speech technology has found niche markets: telephone information systems, access for the disabled, in hands-occupied situations (especially military) and for those suffering RSI. This is discussed in greater detail in Chapter 10, but we can see that it offers three possibilities. The first is as an alternative text entry device to replace the keyboard within an environment and using software originally designed for keyboard use. The second is to redesign a system, taking full advantage of the benefits of the technique whilst minimizing the potential problems. Finally, it can be used in areas where keyboard-based input is impractical or impossible. It is in the latter, more radical areas that speech technology is currently achieving success.

2.3 POSITIONING, POINTING AND DRAWING

Central to most modern computing systems is the ability to point at something on the screen and thereby manipulate it, or perform some function. There has been a long history of such devices, in particular in *computer-aided design* (CAD), where positioning and drawing are the major activities. Pointing devices allow the user to point, position and select items, either directly or by manipulating a pointer on the screen. Many pointing devices can also be used for free-hand drawing although the skill of drawing with a mouse is very different from using a pencil. The mouse is still most common for desktop computers, but is facing challenges as laptop and handheld computing increase their market share. Indeed, these words are being typed on a laptop with a touchpad and no mouse.

2.3.1 The mouse

The mouse has become a major component of the majority of desktop computer systems sold today, and is the little box with the tail connecting it to the machine in our basic computer system picture (Figure 2.1). It is a small, palm-sized box housing a weighted ball – as the box is moved over the tabletop, the ball is rolled by the table and so rotates inside the housing. This rotation is detected by small rollers that are in contact with the ball, and these adjust the values of potentiometers. If you remove the ball occasionally to clear dust you may be able to see these rollers. The changing values of these potentiometers can be directly related to changes in position of the ball. The potentiometers are aligned in different directions so that they can detect both horizontal and vertical motion. The relative motion information is passed to the computer via a wire attached to the box, or in some cases using wireless or infrared, and moves a pointer on the screen, called the *cursor*. The whole arrangement tends to look rodent-like, with the box acting as the body and the wire as the tail; hence the term ‘mouse’. In addition to detecting motion, the mouse has typically one, two or three buttons on top. These are used to indicate selection or to initiate action. Single-button mice tend to have similar functionality to multi-button mice, and achieve this by instituting different operations for a single and a double button click. A ‘double-click’ is when the button is pressed twice in rapid succession. Multi-button mice tend to allocate one operation to each particular button.

The mouse operates in a planar fashion, moving around the desktop, and is an indirect input device, since a transformation is required to map from the horizontal nature of the desktop to the vertical alignment of the screen. Left–right motion is directly mapped, whilst up–down on the screen is achieved by moving the mouse away–towards the user. The mouse only provides information on the relative movement of the ball within the housing: it can be physically lifted up from the desktop and replaced in a different position without moving the cursor. This offers the advantage that less physical space is required for the mouse, but suffers from being less intuitive for novice users. Since the mouse sits on the desk, moving it about is easy and users suffer little arm fatigue, although the indirect nature of the medium can lead to problems with hand–eye coordination. However, a major advantage of the mouse is that the cursor itself is small, and it can be easily manipulated without obscuring the display.

The mouse was developed around 1964 by Douglas C. Engelbart, and a photograph of the first prototype is shown in Figure 2.6. This used two wheels that slid across the desktop and transmitted *x*–*y* coordinates to the computer. The housing was carved in wood, and has been damaged, exposing one of the wheels. The original design actually offers a few advantages over today's more sleek versions: by tilting it so that only one wheel is in contact with the desk, pure vertical or horizontal motion can be obtained. Also, the problem of getting the cursor across the large screens that are often used today can be solved by flicking your wrist to get the horizontal wheel spinning. The mouse pointer then races across the screen with no further effort on your behalf, until you stop it at its destination by dropping the mouse down onto the desktop.

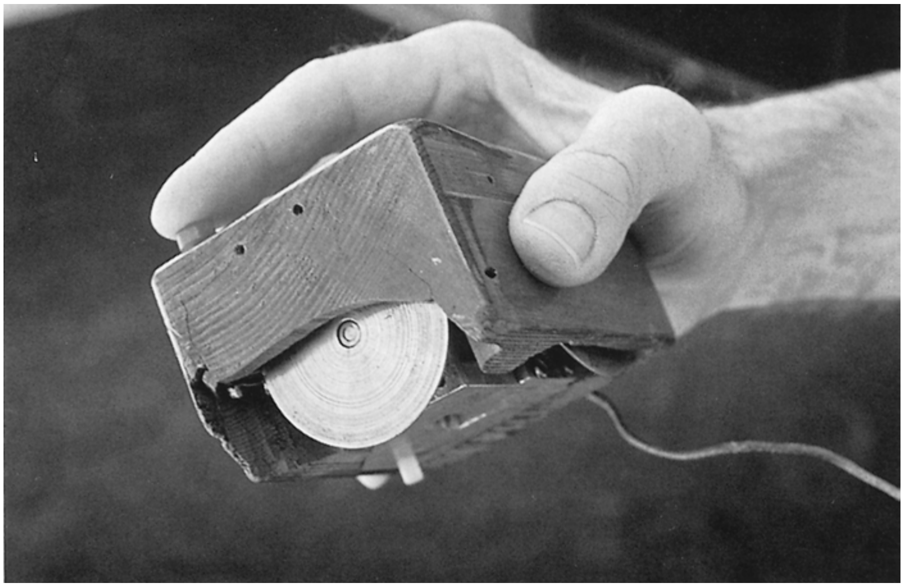


Figure 2.6 The first mouse. Photograph courtesy of Douglas Engelbart and Bootstrap Institute

Optical mice

Optical mice work differently from mechanical mice. A light-emitting diode emits a weak red light from the base of the mouse. This is reflected off a special pad with a metallic grid-like pattern upon which the mouse has to sit, and the fluctuations in reflected intensity as the mouse is moved over the gridlines are recorded by a sensor in the base of the mouse and translated into relative *x*, *y* motion. Some optical mice do not require special mats, just an appropriate surface, and use the natural texture of the surface to detect movement. The optical mouse is less susceptible to dust and dirt than the mechanical one in that its mechanism is less likely to become blocked up. However, for those that rely on a special mat, if the mat is not properly aligned, movement of the mouse may become erratic – especially difficult if you are working with someone and pass the mouse back and forth between you.

Although most mice are hand operated, not all are – there have been experiments with a device called the *footmouse*. As the name implies, it is a foot-operated device, although more akin to an isometric joystick than a mouse. The cursor is moved by foot pressure on one side or the other of a pad. This allows one to dedicate hands to the keyboard. A rare device, the footmouse has not found common acceptance!

Interestingly foot pedals are used heavily in musical instruments including pianos, electric guitars, organs and drums and also in mechanical equipment including cars, cranes, sewing machines and industrial controls. So it is clear that in principle this is a good idea. Two things seem to have limited their use in computer equipment (except simulators and games). One is the practicality of having foot controls in the work environment: pedals under a desk may be operated accidentally, laptops with foot pedals would be plain awkward. The second issue is the kind of control being exercised. Pedals in physical interfaces are used predominantly to control one or more single-dimensional analog controls. It may be that in more specialized interfaces appropriate foot-operated controls could be more commonly and effectively used.

2.3.2 Touchpad

Touchpads are touch-sensitive tablets usually around 2–3 inches (50–75 mm) square. They were first used extensively in Apple Powerbook portable computers but are now used in many other notebook computers and can be obtained separately to replace the mouse on the desktop. They are operated by stroking a finger over their surface, rather like using a simulated trackball. The feel is very different from other input devices, but as with all devices users quickly get used to the action and become proficient.

Because they are small it may require several strokes to move the cursor across the screen. This can be improved by using acceleration settings in the software linking the trackpad movement to the screen movement. Rather than having a fixed ratio of pad distance to screen distance, this varies with the speed of movement. If the finger moves slowly over the pad then the pad movements map to small distances on the screen. If the finger is moving quickly the same distance on the touchpad moves the cursor a long distance. For example, on the trackpad being used when writing this section a very slow movement of the finger from one side of the trackpad to the other moves the cursor less than 10% of the width of the screen. However, if the finger is moved very rapidly from side to side, the cursor moves the whole width of the screen.

In fact, this form of acceleration setting is also used in other indirect positioning devices including the mouse. Fine settings of this sort of parameter makes a great difference to the 'feel' of the device.

2.3.3 Trackball and thumbwheel

The trackball is really just an upside-down mouse! A weighted ball faces upwards and is rotated inside a static housing, the motion being detected in the same way as for a mechanical mouse, and the relative motion of the ball moves the cursor. Because of this, the trackball requires no additional space in which to operate, and is therefore a very compact device. It is an indirect device, and requires separate buttons for selection. It is fairly accurate, but is hard to draw with, as long movements are difficult. Trackballs now appear in a wide variety of sizes, the most usual being about the same as a golf ball, with a number of larger and smaller devices available. The size and 'feel' of the trackball itself affords significant differences in the usability of the device: its weight, rolling resistance and texture all contribute to the overall effect.

Some of the smaller devices have been used in notebook and portable computers, but more commonly trackpads or nipples are used. They are often sold as alternatives to mice on desktop computers, especially for RSI sufferers. They are also heavily used in video games where their highly responsive behavior, including being able to spin the ball, is ideally suited to the demands of play.

Thumbwheels are different in that they have two orthogonal dials to control the cursor position. Such a device is very cheap, but slow, and it is difficult to manipulate the cursor in any way other than horizontally or vertically. This limitation can sometimes be a useful constraint in the right application. For instance, in CAD the designer is almost always concerned with exact verticals and horizontals, and a device that provides such constraints is very useful, which accounts for the appearance of thumbwheels in CAD systems. Another successful application for such a device has been in a drawing game such as Etch-a-Sketch in which straight lines can be created on a simple screen, since the predominance of straight lines in simple drawings means that the motion restrictions are an advantage rather than a handicap. However, if you were to try to write your signature using a thumbwheel, the limitations would be all too apparent. The appropriateness of the device depends on the task to be performed.

Although two-axis thumbwheels are not heavily used in mainstream applications, single thumbwheels are often included on a standard mouse in order to offer an alternative means to scroll documents. Normally scrolling requires you to grab the scroll bar with the mouse cursor and drag it down. For large documents it is hard to be accurate and in addition the mouse dragging is done holding a finger down which adds to hand strain. In contrast the small scroll wheel allows comparatively intuitive and fast scrolling, simply rotating the wheel to move the page.

2.3.4 Joystick and keyboard nipple

The joystick is an indirect input device, taking up very little space. Consisting of a small palm-sized box with a stick or shaped grip sticking up from it, the joystick is a simple device with which movements of the stick cause a corresponding movement of the screen cursor. There are two types of joystick: the *absolute* and the *isometric*. In the absolute joystick, movement is the important characteristic, since the position of the joystick in the base corresponds to the position of the cursor on the screen. In the isometric joystick, the pressure on the stick corresponds to the velocity of the cursor, and when released, the stick returns to its usual upright centered position. This type of joystick is also called the velocity-controlled joystick, for obvious reasons. The buttons are usually placed on the top of the stick, or on the front like a trigger. Joysticks are inexpensive and fairly robust, and for this reason they are often found in computer games. Another reason for their dominance of the games market is their relative familiarity to users, and their likeness to aircraft joysticks: aircraft are a favorite basis for games, leading to familiarity with the joystick that can be used for more obscure entertainment ideas.

A smaller device but with the same basic characteristics is used on many laptop computers to control the cursor. Some older systems had a variant of this called the keymouse, which was a single key. More commonly a small rubber nipple projects in the center of the keyboard and acts as a tiny isometric joystick. It is usually difficult for novices to use, but this seems to be related to fine adjustment of the speed settings. Like the joystick the nipple controls the rate of movement across the screen and is thus less direct than a mouse or stylus.

2.3.5 Touch-sensitive screens (touchscreens)

Touchscreens are another method of allowing the user to point and select objects on the screen, but they are much more direct than the mouse, as they detect the presence of the user's finger, or a stylus, on the screen itself. They work in one of a number of different ways: by the finger (or stylus) interrupting a matrix of light beams, or by capacitance changes on a grid overlaying the screen, or by ultrasonic reflections. Because the user indicates exactly which item is required by pointing to it, no mapping is required and therefore this is a direct device.

The touchscreen is very fast, and requires no specialized pointing device. It is especially good for selecting items from menus displayed on the screen. Because the screen acts as an input device as well as an output device, there is no separate hardware to become damaged or destroyed by dirt; this makes touchscreens suitable for use in hostile environments. They are also relatively intuitive to use and have been used successfully as an interface to information systems for the general public.

They suffer from a number of disadvantages, however. Using the finger to point is not always suitable, as it can leave greasy marks on the screen, and, being a fairly blunt instrument, it is quite inaccurate. This means that the selection of small regions is very difficult, as is accurate drawing. Moreover, lifting the arm to point to a vertical screen is very tiring, and also means that the screen has to be within about a meter of the user to enable it to be reached, which can make it too close for comfort. Research has shown that the optimal angle for a touchscreen is about 15 degrees up from the horizontal.

2.3.6 Stylus and light pen

For more accurate positioning (and to avoid greasy screens), systems with touch-sensitive surfaces often emply a stylus. Instead of pointing at the screen directly a small pen-like plastic stick is used to point and draw on the screen. This is particularly popular in PDAs, but they are also being used in some laptop computers.

An older technology that is used in the same way is the light pen. The pen is connected to the screen by a cable and, in operation, is held to the screen and detects a burst of light from the screen phosphor during the display scan. The light pen can therefore address individual pixels and so is much more accurate than the touchscreen.

Both stylus and light pen can be used for fine selection and drawing, but both can be tiring to use on upright displays and are harder to take up and put down when used together with a keyboard. Interestingly some users of PDAs with fold-out keyboards learn to hold the stylus held outwards between their fingers so that they can type whilst holding it. As it is unattached the stylus can easily get lost, but a closed pen can be used in emergencies.

Stylus, light pen and touchscreen are all very direct in that the relationship between the device and the thing selected is immediate. In contrast, mouse, touchpad, joystick and trackball all have to map movements on the desk to cursor movement on the screen.

However, the direct devices suffer from the problem that, in use, the act of pointing actually obscures the display, making it harder to use, especially if complex detailed selections or movements are required in rapid succession. This means that screen designs have to take into account where the user's hand will be. For example, you may want to place menus at the bottom of the screen rather than the top. Also you may want to offer alternative layouts for right-handed and left-handed users.

2.3.7 Digitizing tablet

The digitizing tablet is a more specialized device typically used for freehand drawing, but may also be used as a mouse substitute. Some highly accurate tablets, usually using a puck (a mouse-like device), are used in special applications such as digitizing information for maps.

The tablet provides positional information by measuring the position of some device on a special pad, or *tablet*, and can work in a number of ways. The *resistive* *tablet* detects point contact between two separated conducting sheets. It has advantages in that it can be operated without a specialized stylus – a pen or the user's finger is sufficient. The *magnetic tablet* detects current pulses in a magnetic field using a small loop coil housed in a special pen. There are also capacitative and electrostatic tablets that work in a similar way. The *sonic tablet* is similar to the above but requires no special surface. An ultrasonic pulse is emitted by a special pen which is detected by two or more microphones which then triangulate the pen position. This device can be adapted to provide 3D input, if required.

Digitizing tablets are capable of high resolution, and are available in a range of sizes. Sampling rates vary, affecting the resolution of cursor movement, which gets progressively finer as the sampling rate increases. The digitizing tablet can be used to detect relative motion *or* absolute motion, but is an indirect device since there is a mapping from the plane of operation of the tablet to the screen. It can also be used for text input; if supported by character recognition software, handwriting can be interpreted. Problems with digitizing tablets are that they require a large amount of desk space, and may be awkward to use if displaced to one side by the keyboard.

2.3.8 Eyegaze

Eyegaze systems allow you to control the computer by simply looking at it! Some systems require you to wear special glasses or a small head-mounted box, others are built into the screen or sit as a small box below the screen. A low-power laser is shone into the eye and is reflected off the retina. The reflection changes as the angle of the eye alters, and by tracking the reflected beam the eyegaze system can determine the direction in which the eye is looking. The system needs to be calibrated, typically by staring at a series of dots on the screen, but thereafter can be used to move the screen cursor or for other more specialized uses. Eyegaze is a very fast and accurate device, but the more accurate versions can be expensive. It is fine for selection but not for drawing since the eye does not move in smooth lines. Also in real applications it can be difficult to distinguish deliberately gazing at something and accidentally glancing at it.

Such systems have been used in military applications, notably for guiding air-to-air missiles to their targets, but are starting to find more peaceable uses, for disabled users and for workers in environments where it is impossible for them to use their hands. The rarity of the eyegaze is due partly to its novelty and partly to its expense, and it is usually found only in certain domain-specific applications. Within HCI it is particularly useful as part of evaluation as one is able to trace exactly where the user is looking [81]. As prices drop and the technology becomes less intrusive we may see more applications using eyegaze, especially in virtual reality and augmented reality areas (see Chapter 20).

2.3.9 Cursor keys and discrete positioning

All of the devices we have discussed are capable of giving near continuous 2D positioning, with varying degrees of accuracy. For many applications we are only

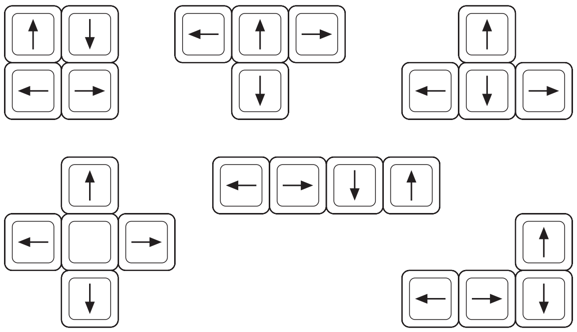


Figure 2.7 Various cursor key layouts

interested in positioning within a sequential list such as a menu or amongst 2D cells as in a spreadsheet. Even for moving within text discrete up/down left/right keys can sometimes be preferable to using a mouse.

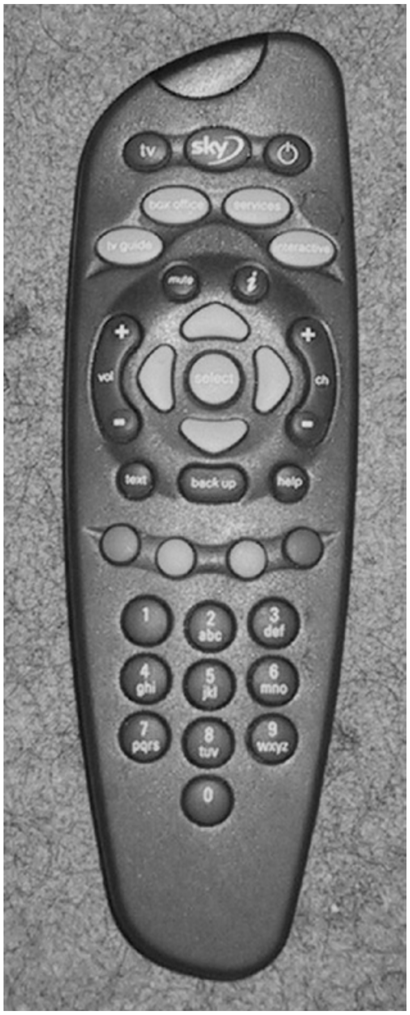
Cursor keys are available on most keyboards. Four keys on the keyboard are used to control the cursor, one each for up, down, left and right. There is no standardized layout for the keys. Some layouts are shown in Figure 2.7, but the most common now is the inverted 'T'.

Cursor keys used to be more heavily used in character-based systems before windows and mice were the norm. However, when logging into remote machines such as web servers, the interface is often a virtual character-based terminal within a telnet window. In such applications it is common to find yourself in a 1970s world of text editors controlled sometimes using cursor keys and sometimes by more arcane combinations of control keys!

Small devices such as mobile phones, personal entertainment and television remote controls often require discrete control, either dedicated to a particular function such as volume, or for use as general menu selection. Figure 2.8 shows examples of these. The satellite TV remote control has dedicated '+/–' buttons for controlling volume and stepping between channels. It also has a central cursor pad that is used for on-screen menus. The mobile phone has a single central joystick-like device. This can be pushed left/right, up/down to navigate within the small 3×3 array of graphical icons as well as select from text menus.

2.4 DISPLAY DEVICES

The vast majority of interactive computer systems would be unthinkable without some sort of display screen, but many such systems do exist, though usually in specialized applications only. Thinking beyond the traditional, systems such as cars, hi-fis and security alarms all have different outputs from those expressible on a screen, but in the personal computer and workstation market, screens are pervasive.



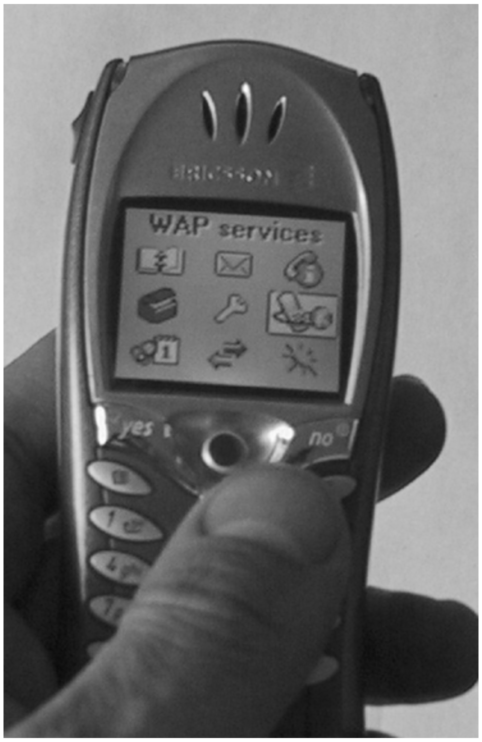


Figure 2.8 Satellite TV remote control and mobile phone. Source: Photograph left by Alan Dix with permission from British Sky Broadcasting Limited, photograph right by Alan Dix (Ericsson phone)

In this section, we discuss the standard computer display in detail, looking at the properties of bitmap screens, at different screen technologies, at large and situated displays, and at a new technology, 'digital paper'.

2.4.1 Bitmap displays – resolution and color

Virtually all computer displays are based on some sort of bitmap. That is the display is made of vast numbers of colored dots or pixels in a rectangular grid. These pixels may be limited to black and white (for example, the small display on many TV remote controls), in grayscale, or full color.

The color or, for monochrome screens, the intensity at each pixel is held by the computer's video card. One bit per pixel can store on/off information, and hence only black and white (the term 'bitmap' dates from such displays). More bits per pixel give rise to more color or intensity possibilities. For example, 8 bits/pixel give rise to $2^8 = 256$ possible colors *at any one time*. The set of colors make up what is called the *colormap*, and the colormap can be altered at any time to produce a different set of colors. The system is therefore capable of actually displaying many more than the number of colors in the colormap, but not simultaneously. Most desktop computers now use 24 or 32 bits per pixel which allows virtually unlimited colors, but devices such as mobile phones and PDAs are often still monochrome or have limited color range.

As well as the number of colors that can be displayed at each pixel, the other measure that is important is the resolution of the screen. Actually the word 'resolution' is used in a confused (and confusing!) way for screens. There are two numbers to consider:

- the *total number* of pixels: in standard computer displays this is always in a 4:3 ratio, perhaps 1024 pixels across by 768 down, or 1600×1200; for PDAs this will be more in the order of a few hundred pixels in each direction.
- the *density* of pixels: this is measured in pixels per inch. Unlike printers (see Section 2.7 below) this density varies little between 72 and 96 pixels per inch.

To add to the confusion, a monitor, liquid crystal display (LCD) screen or other display device will quote its maximum resolution, but the computer may actually give it less than this. For example, the screen may be a 1200×900 resolution with 96 pixels per inch, but the computer only sends it 800×600. In the case of a cathode ray tube (CRT) this typically will mean that the image is stretched over the screen surface giving a lower density of 64 pixels per inch. An LCD screen cannot change its pixel size so it would keep 96 pixels per inch and simply not use all its screen space, adding a black border instead. Some LCD projectors will try to stretch or reduce what they are given, but this may mean that one pixel gets stretched to two, or two pixels get 'squashed' into one, giving rise to display 'artifacts' such as thin lines disappearing, or uniform lines becoming alternately thick or thin.

Although horizontal and vertical lines can be drawn perfectly on bitmap screens, and lines at 45 degrees reproduce reasonably well, lines at any other angle and curves have 'jaggies', rough edges caused by the attempt to approximate the line with pixels.

When using a single color jaggies are inevitable. Similar effects are seen in bitmap fonts. The problem of jaggies can be reduced by using high-resolution screens, or by a technique known as *anti-aliasing*. Anti-aliasing softens the edges of line segments, blurring the discontinuity and making the jaggies less obvious.

Look at the two images in Figure 2.9 with your eyes slightly screwed up. See how the second anti-aliased line looks better. Of course, screen resolution is much higher, but the same principle holds true. The reason this works is because our brains are constantly 'improving' what we see in the world: processing and manipulating the raw sensations of the rods and cones in our eyes and turning them into something meaningful. Often our vision is blurred because of poor light, things being out of focus, or defects in our vision. Our brain compensates and tidies up blurred images. By deliberately blurring the image, anti-aliasing triggers this processing in our brain and we appear to see a smooth line at an angle.

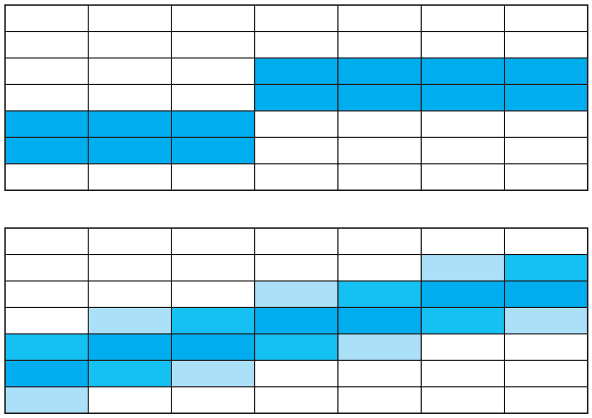


Figure 2.9 Magnified anti-aliased lines

2.4.2 Technologies

Cathode ray tube

The cathode ray tube is the television-like computer screen still most common as we write this, but rapidly being displaced by flat LCD screens. It works in a similar way to a standard television screen. A stream of electrons is emitted from an electron gun, which is then focussed and directed by magnetic fields. As the beam hits the phosphor-coated screen, the phosphor is excited by the electrons and glows (see Figure 2.10). The electron beam is scanned from left to right, and then flicked back to rescan the next line, from top to bottom. This is repeated, at about 30 Hz (that is, 30 times a second), per frame, although higher scan rates are sometimes used to reduce the flicker on the screen. Another way of reducing flicker is to use *interlacing*, in which the odd lines on the screen are all scanned first, followed by the even lines. Using a high-persistence phosphor, which glows for a longer time when excited, also reduces flicker, but causes image smearing especially if there is significant animation.

Black and white screens are able to display grayscale by varying the intensity of the electron beam; color is achieved using more complex means. Three electron guns are used, one each to hit red, green and blue phosphors. Combining these colors can

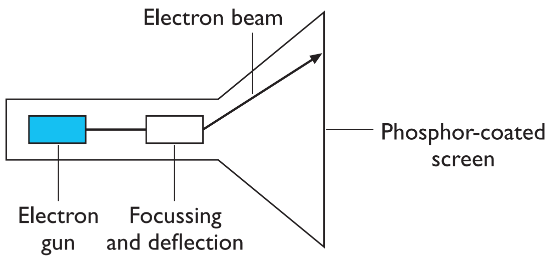


Figure 2.10 CRT screen

produce many others, including white, when they are all fully on. These three phosphor dots are focussed to make a single point using a *shadow mask*, which is imprecise and gives color screens a lower resolution than equivalent monochrome screens.

An alternative approach to producing color on the screen is to use *beam penetration*. A special phosphor glows a different color depending on the intensity of the beam hitting it.

The CRT is a cheap display device and has fast enough response times for rapid animation coupled with a high color capability. Note that animation does not necessarily mean little creatures and figures running about on the screen, but refers in a more general sense to the use of motion in displays: moving the cursor, opening windows, indicating processor-intensive calculations, or whatever. As screen resolution increases, however, the price rises. Because of the electron gun and focussing components behind the screen, CRTs are fairly bulky, though recent innovations have led to flatter displays in which the electron gun is not placed so that it fires directly at the screen, but fires parallel to the screen plane with the resulting beam bent through 90 degrees to hit the screen.

Health hazards of CRT displays

Most people who habitually use computers are aware that screens can often cause eyestrain and fatigue; this is usually due to flicker, poor legibility or low contrast. There have also been many concerns relating to the emission of radiation from screens. These can be categorized as follows:

- X-rays which are largely absorbed by the screen (but not at the rear!)
- ultraviolet and infrared radiation from phosphors in insignificant levels
- radio frequency emissions, plus ultrasound (approximately 16 kHz)
- electrostatic field which leaks out through the tube to the user. The intensity is dependent on distance and humidity. This can cause rashes in the user
- electromagnetic fields (50 Hz to 0.5 MHz) which create induction currents in conductive materials, including the human body. Two types of effects are attributed to this: in the visual system, a high incidence of cataracts in visual display unit (VDU) operators, and concern over reproductive disorders (miscarriages and birth defects).

Research into the potentially harmful effect of these emissions is generally inconclusive, in that it is difficult to determine precisely what the causes of illness are, and many health scares have been the result of misinformed media opinion rather than scientific fact. However, users who are pregnant ought to take especial care and observe simple precautions. Generally, there are a number of common-sense things that can be done to relieve strain and minimize any risk. These include

- not sitting too close to the screen
- not using very small fonts
- not looking at the screen for a long time without a break
- working in well-lit surroundings
- not placing the screen directly in front of a bright window.

Liquid crystal display

If you have used a personal organizer or notebook computer, you will have seen the light, flat plastic screens. These displays utilize liquid crystal technology and are smaller, lighter and consume far less power than traditional CRTs. These are also commonly referred to as flat-panel displays. They have no radiation problems associated with them, and are matrix addressable, which means that individual pixels can be accessed without the need for scanning.

Similar in principle to the digital watch, a thin layer of liquid crystal is sandwiched between two glass plates. The top plate is transparent and polarized, whilst the bottom plate is reflective. External light passes through the top plate and is polarized, which means that it only oscillates in one direction. This then passes through the crystal, reflects off the bottom plate and back to the eye, and so that cell looks white. When a voltage is applied to the crystal, via the conducting glass plates, the crystal twists. This causes it to turn the plane of polarization of the incoming light, rotating it so that it cannot return through the top plate, making the activated cell look black. The LCD requires refreshing at the usual rates, but the relatively slow response of the crystal means that flicker is not usually noticeable. The low intensity of the light emitted from the screen, coupled with the reduced flicker, means that the LCD is less tiring to use than standard CRT ones, with reduced eyestrain.

This different technology can be used to replace the standard screen on a desktop computer, and this is now common. However, the particular characteristics of compactness, light weight and low power consumption have meant that these screens have created a large niche in the computer market by monopolizing the notebook and portable computer systems side. The advent of these screens allowed small, light computers to be built, and created a large market that did not previously exist. Such computers, riding on the back of the technological wave, have opened up a different way of working for many people, who now have access to computers when away from the office, whether out on business or at home. Working in a different location on a smaller machine with different software obviously represents a different style of interaction and so once again we can see that differences in devices may alter the human–computer interaction considerably. The growing notebook computer market fed back into an investment in developing LCD screen technology, with supertwisted crystals increasing the viewing angle dramatically. Response times have also improved so that LCD screens are now used in personal DVD players and even in home television.

When the second edition of this book was being written the majority of LCD screens were black and white or grayscale, We wrote then ‘it will be interesting to see whether color LCD screens supersede grayscale by the time the third edition of this book is prepared’. Of course, this is precisely the case. Our expectation is that by the time we produce the next edition LCD monitors will have taken over from CRT monitors completely.

Special displays

There are a number of other display technologies used in niche markets. The one you are most likely to see is the gas plasma display, which is used in large screens (see Section 2.4.3 below).

The random scan display, also known as the *directed beam refresh*, or *vector display*, works differently from the bitmap display, also known as raster scan, that we discussed in Section 2.4.1. Instead of scanning the whole screen sequentially and horizontally, the random scan draws the lines to be displayed directly. By updating the screen at at least 30 Hz to reduce flicker, the direct drawing of lines at any angle means that jaggies are not created, and higher resolutions are possible, up to 4096 × 4096 pixels. Color on such displays is achieved using beam penetration technology, and is generally of a poorer quality. Eyestrain and fatigue are still a problem, and these displays are more expensive than raster scan ones, so they are now only used in niche applications.

The *direct view storage tube* is used extensively as the display for an analog storage oscilloscope, which is probably the only place that these displays are used in any great numbers. They are similar in operation to the random scan CRT but the image is maintained by flood guns which have the advantage of producing a stable display with no flicker. The screen image can be incrementally updated but not selectively erased; removing items has to be done by redrawing the new image on a completely erased screen. The screens have a high resolution, typically about 4096 × 3120 pixels, but suffer from low contrast, low brightness and a difficulty in displaying color.

2.4.3 Large displays and situated displays

Displays are no longer just things you have on your desktop or laptop. In Chapter 19 we will discuss meeting room environments that often depend on large shared screens. You may have attended lectures where the slides are projected from a computer onto a large screen. In shops and garages large screen adverts assault us from all sides.

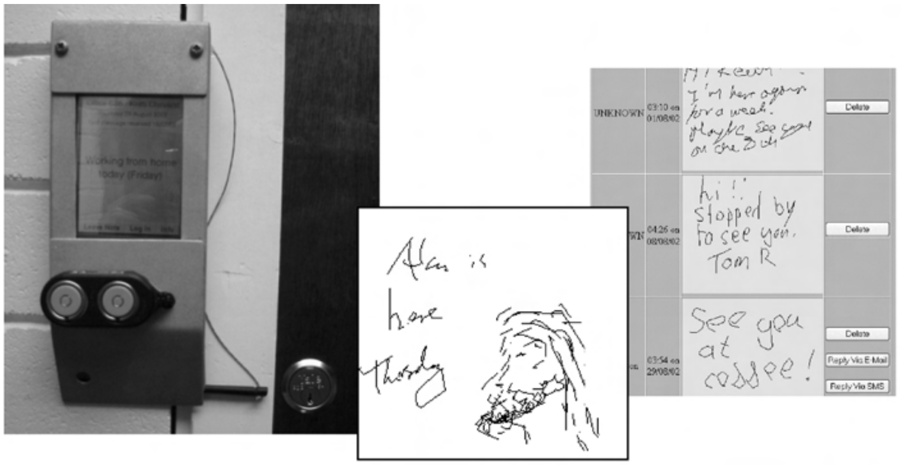
There are several types of large screen display. Some use gas plasma technology to create large flat bitmap displays. These behave just like a normal screen except they are big and usually have the HDTV (high definition television) wide screen format which has an aspect ratio of 16:9 instead of the 4:3 on traditional TV and monitors.

Where very large screen areas are required, several smaller screens, either LCD or CRT, can be placed together in a video wall. These can display separate images, or a single TV or computer image can be split up by software or hardware so that each screen displays a portion of the whole and the result is an enormous image. This is the technique often used in large concerts to display the artists or video images during the performance.

Possibly the large display you are most likely to have encountered is some sort of projector. There are two variants of these. In very large lecture theatres, especially older ones, you see projectors with large red, green and blue lenses. These each scan light across the screen to build a full color image. In smaller lecture theatres and in small meetings you are likely to see LCD projectors. Usually the size of a large book, these are like ordinary slide projectors except that where the slide would be there is a small LCD screen instead. The light from the projector passes through the tiny screen and is then focussed by the lens onto the screen.

The disadvantage of projected displays is that the presenter's shadow can often fall across the screen. Sometimes this is avoided in fixed lecture halls by using back projection. In a small room behind the screen of the lecture theatre there is a projector producing a right/left reversed image. The screen itself is a semi-frosted glass so that the image projected on the back can be seen in the lecture theatre. Because there are limits on how wide an angle the projector can manage without distortion, the size of the image is limited by the depth of the projection room behind, so these are less heavily used than front projection.

As well as for lectures and meetings, display screens can be used in various public places to offer information, link spaces or act as message areas. These are often called *situated displays* as they take their meaning from the location in which they are situated. These may be large screens where several people are expected to view or interact simultaneously, or they may be very small. Figure 2.11 shows an example of a small experimental situated display mounted by an office door to act as an electronic sticky note [70].



(i) Door display (ii) Occupant's note (iii) Web interface

Figure 2.11 Situated door display. Source: Courtesy of Keith Cheverst

Design Focus

Hermes: a situated display

Office doors are often used as a noticeboard with messages from the occupant such as 'just gone out' or 'timetable for the week' and from visitors 'missed you, call when you get back'. The Hermes system is an electronic door display that offers some of the functions of sticky notes on a door [70]. Figure 2.11(i) shows an installed Hermes device fixed just beside the door, including the socket to use a Java iButton to authenticate the occupant. The occupant can leave messages that others can read (Figure 2.11(ii)) and people coming to the door can leave messages for the occupant. Electronic notes are smaller than paper ones, but because they are electronic they can be read remotely using a web interface (Figure 2.11(iii)), or added by SMS (see Chapter 19, Section 19.3.2).

The fact that it is situated – by a person's door – is very important. It establishes a context, 'Alan's door', and influences the way the system is used. For example, the idea of anonymous messages left on the door, where the visitor has had to be physically present, feels different from, say, anonymous emails.

See the book website for the full case study: /e3/casestudy/hermes/

2.4.4 Digital paper

A new form of 'display' that is still in its infancy is the various forms of digital paper. These are thin flexible materials that can be written to electronically, just like a computer screen, but which keep their contents even when removed from any electrical supply.

There are various technologies being investigated for this. One involves the whole surface being covered with tiny spheres, black one side, white the other. Electronics embedded into the material allow each tiny sphere to be rotated to make it black or white. When the electronic signal is removed the ball stays in its last orientation. A different technique has tiny tubes laid side by side. In each tube is light-absorbing liquid and a small reflective sphere. The sphere can be made to move to the top surface or away from it making the pixel white or black. Again the sphere stays in its last position once the electronic signal is removed.

Probably the first uses of these will be for large banners that can be reprogrammed or slowly animated. This is an ideal application, as it does not require very rapid updates and does not require the pixels to be small. As the technology matures, the aim is to have programmable sheets of paper that you attach to your computer to get a 'soft' printout that can later be changed. Perhaps one day you may be able to have a 'soft' book that appears just like a current book with soft pages that can be turned and skimmed, but where the contents and cover can be changed when you decide to download a new book from the net!

2.5 DEVICES FOR VIRTUAL REALITY AND 3D INTERACTION

Virtual reality (VR) systems and various forms of 3D visualization are discussed in detail in Chapter 20. These require you to navigate and interact in a three-dimensional space. Sometimes these use the ordinary controls and displays of a desktop computer system, but there are also special devices used both to move and interact with 3D objects and to enable you to see a 3D environment.

2.5.1 Positioning in 3D space

Virtual reality systems present a 3D virtual world. Users need to navigate through these spaces and manipulate the virtual objects they find there. Navigation is not simply a matter of moving to a particular location, but also of choosing a particular orientation. In addition, when you grab an object in real space, you don't simply move it around, but also twist and turn it, for example when opening a door. Thus the move from mice to 3D devices usually involves a change from two degrees of freedom to six degrees of freedom, not just three.

Cockpit and virtual controls

Helicopter and aircraft pilots already have to navigate in real space. Many arcade games and also more serious applications use controls modeled on an aircraft cockpit to 'fly' through virtual space. However, helicopter pilots are very skilled and it takes a lot of practice for users to be able to work easily in such environments.

In many PC games and *desktop virtual reality* (where the output is shown on an ordinary computer screen), the controls are themselves virtual. This may be a simulated form of the cockpit controls or more prosaic up/down left/right buttons. The user manipulates these virtual controls using an ordinary mouse (or other 2D device). Note that this means there are two levels of indirection. It is a tribute to the flexibility of the human mind that people can not only use such systems but also rapidly become proficient.

The 3D mouse

There are a variety of devices that act as 3D versions of a mouse. Rather than just moving the mouse on a tabletop, you can pick it up, move it in three dimensions, rotate the mouse and tip it forward and backward. The 3D mouse has a full six degrees of freedom as its position can be tracked (three degrees), and also its up/down angle (called *pitch*), its left/right orientation (called *yaw*) and the amount it is twisted about its own axis (called *roll*) (see Figure 2.12). Various sensors are used to track the mouse position and orientation: magnetic coils, ultrasound or even mechanical joints where the mouse is mounted rather like an angle-poise lamp.

With the 3D mouse, and indeed most 3D positioning devices, users may experience strain from having to hold the mouse in the air for a long period. Putting the

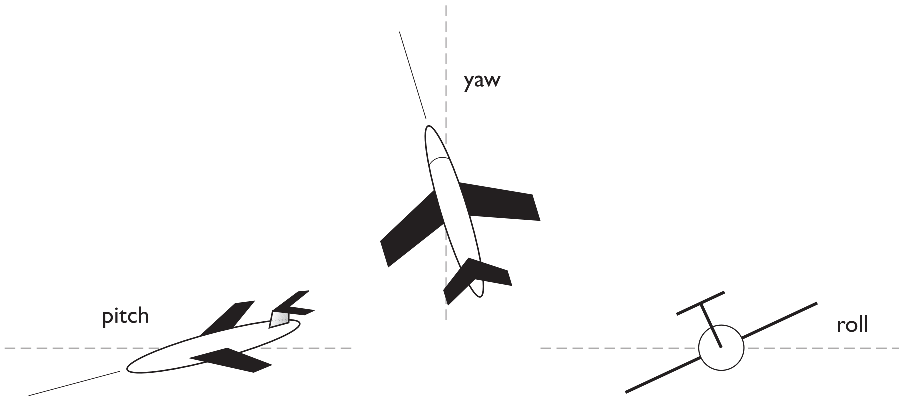


Figure 2.12 Pitch, yaw and roll

3D mouse down may even be treated as an action in the virtual environment, that is taking a nose dive.

Dataglove

One of the mainstays of high-end VR systems (see Chapter 20), the dataglove is a 3D input device. Consisting of a lycra glove with optical fibers laid along the fingers, it detects the joint angles of the fingers and thumb. As the fingers are bent, the fiber optic cable bends too; increasing bend causes more light to leak from the fiber, and the reduction in intensity is detected by the glove and related to the degree of bend in the joint. Attached to the top of the glove are two sensors that use ultrasound to determine 3D positional information as well as the angle of roll, that is the degree of wrist rotation. Such rich multi-dimensional input is currently a solution in search of a problem, in that most of the applications in use do not require such a comprehensive form of data input, whilst those that do cannot afford it. However, the availability of cheaper versions of the dataglove will encourage the development of more complex systems that are able to utilize the full power of the dataglove as an input device. There are a number of potential uses for this technology to assist disabled people, but cost remains the limiting factor at present.

The dataglove has the advantage that it is very easy to use, and is potentially very powerful and expressive (it can provide 10 joint angles, plus the 3D spatial information and degree of wrist rotation, 50 times a second). It suffers from extreme expense, and the fact that it is difficult to use in conjunction with a keyboard. However, such a limitation is shortsighted; one can imagine a keyboard drawn onto a desk, with software detecting hand positions and interpreting whether the virtual keys had been hit or not. The potential for the dataglove is vast; gesture recognition and sign language interpretation are two obvious areas that are the focus of active research, whilst less obvious applications are evolving all the time.

Virtual reality helmets

The helmets or goggles worn in some VR systems have two purposes: (i) they display the 3D world to each eye and (ii) they allow the user's head position to be tracked. We will discuss the former later when we consider output devices. The head tracking is used primarily to feed into the output side. As the user's head moves around the user ought to see different parts of the scene. However, some systems also use the user's head direction to determine the direction of movement within the space and even which objects to manipulate (rather like the eyegaze systems). You can think of this rather like leading a horse in reverse. If you want a horse to go in a particular direction, you use the reins to pull its head in the desired direction and the horse follows its head.

Whole-body tracking

Some VR systems aim to be immersive, that is to make the users feel as if they are really in the virtual world. In the real world it is possible (although not usually wise) to walk without looking in the direction you are going. If you are driving down the road and glance at something on the roadside you do not want the car to do a sudden 90-degree turn! Some VR systems therefore attempt to track different kinds of body movement. Some arcade games have a motorbike body on which you can lean into curves. More strangely, small trampolines have been wired up so that the user can control movement in virtual space by putting weight on different parts of the trampoline. The user can literally surf through virtual space. In the extreme the movement of the whole body may be tracked using devices similar to the dataglove, or using image-processing techniques. In the latter, white spots are stuck at various points of the user's body and the position of these tracked using two or more cameras, allowing the location of every joint to be mapped. Although the last of these sounds a little constraining for the fashion conscious it does point the way to less intrusive tracking techniques.

2.5.2 3D displays

Just as the 3D images used in VR have led to new forms of input device, they also require more sophisticated outputs. Desktop VR is delivered using a standard computer screen and a 3D impression is produced by using effects such as shadows, occlusion (where one object covers another) and perspective. This can be very effective and you can even view 3D images over the world wide web using a VRML (virtual reality markup language) enabled browser.

Seeing in 3D

Our eyes use many cues to perceive depth in the real world (see also Chapter 1). It is in fact quite remarkable as each eye sees only a flattened form of the world, like a photograph. One important effect is *stereoscopic vision* (or simply *stereo vision*).

Because each eye is looking at an object from a slightly different angle each sees a different image and our brain is able to use this to assess the relative distance of different objects. In desktop VR this stereoscopic effect is absent. However, various devices exist to deliver true stereoscopic images.

The start point of any stereoscopic device is the generation of images from different perspectives. As the computer is generating images for the virtual world anyway, this just means working out the right positions and angles corresponding to the typical distance between eyes on a human face. If this distance is too far from the natural one, the user will be presented with a giant's or gnat's eye view of the world!

Different techniques are then used to ensure that each eye sees the appropriate image. One method is to have two small screens fitted to a pair of goggles. A different image is then shown to each eye. These devices are currently still quite cumbersome and the popular image of VR is of a user with head encased in a helmet with something like a pair of inverted binoculars sticking out in front. However, smaller and lighter LCDs are now making it possible to reduce the devices towards the size and weight of ordinary spectacles.

An alternative method is to have a pair of special spectacles connected so that each eye can be blanked out by timed electrical signals. If this is synchronized with the frame rate of a computer monitor, each eye sees alternate images. Similar techniques use polarized filters in front of the monitor and spectacles with different polarized lenses. These techniques are both effectively using similar methods to the red–green 3D spectacles given away in some breakfast cereals. Indeed, these red–green spectacles have been used in experiments in wide-scale 3D television broadcasts. However, the quality of the 3D image from the polarized and blanked eye spectacles is substantially better.

The ideal would be to be able to look at a special 3D screen and see 3D images just as one does with a hologram – 3D television just like in all the best sci-fi movies! But there is no good solution to this yet. One method is to inscribe the screen with small vertical grooves forming hundreds of prisms. Each eye then sees only alternate dots on the screen allowing a stereo image at half the normal horizontal resolution. However, these screens have very narrow *viewing angles*, and are not ready yet for family viewing.

In fact, getting stereo images is not the whole story. Not only do our eyes see different things, but each eye also focusses on the current object of interest (small muscles change the shape of the lens in the pupil of the eye). The images presented to the eye are generated at some fixed focus, often with effectively infinite *depth of field*. This can be confusing and tiring. There has been some progress recently on using lasers to detect the focal depth of each eye and adjust the images correspondingly, similar to the technology used for eye tracking. However, this is not currently used extensively.

VR motion sickness

We all get annoyed when computers take a long time to change the screen, pop up a window, or play a digital movie. However, with VR the effects of poor display performance can be more serious. In real life when we move our head the image our eyes see changes accordingly. VR systems produce the same effect by using sensors in the goggles or helmet and then using the position of the head to determine the right image to show. If the system is slow in producing these images a lag develops between the user moving his head and the scene changing. If this delay is more than a hundred milliseconds or so the feeling becomes disorienting. The effect is very similar to that of being at sea. You stand on the deck looking out to sea, the boat gently rocking below you. Tiny channels in your ears detect the movement telling your brain that you are moving; your eyes see the horizon moving in one direction and the boat in another. Your brain gets confused and you get sick. Users of VR can experience similar nausea and few can stand it for more than a short while. In fact, keeping laboratories sanitary has been a major push in improving VR technology.

Simulators and VR caves

Because of the problems of delivering a full 3D environment via head-mounted displays, some virtual reality systems work by putting the user within an environment where the virtual world is displayed upon it. The most obvious examples of this are large flight simulators – you go inside a mock-up of an aircraft cockpit and the scenes you would see through the windows are projected onto the virtual windows. In motorbike or skiing simulators in video arcades large screens are positioned to fill the main part of your visual field. You can still look over your shoulder and see your friends, but while you are engaged in the game it surrounds you.

More general-purpose rooms called caves have large displays positioned all around the user, or several back projectors. In these systems the user can look all around and see the virtual world surrounding them.

2.6 PHYSICAL CONTROLS, SENSORS AND SPECIAL DEVICES

As we have discussed, computers are coming out of the box. The mouse keyboard and screen of the traditional computer system are not relevant or possible in applications that now employ computers such as interactive TV, in-car navigation systems or personal entertainment. These devices may have special displays, may use sound, touch and smell as well as visual displays, may have dedicated controls and may sense the environment or your own bio-signs.

2.6.1 Special displays

Apart from the CRT screen there are a number of visual outputs utilized in complex systems, especially in embedded systems. These can take the form of analog representations of numerical values, such as dials, gauges or lights to signify a certain system state. Flashing light-emitting diodes (LEDs) are used on the back of some computers to signify the processor state, whilst gauges and dials are found in process control systems. Once you start in this mode of thinking, you can contemplate numerous visual outputs that are unrelated to the screen. One visual display that has found a specialized niche is the head-up display that is used in aircraft. The pilot is fully occupied looking forward and finds it difficult to look around the cockpit to get information. There are many different things that need to be known, ranging from data from tactical systems to navigational information and aircraft status indicators. The head-up display projects a subset of this information into the pilot's line of vision so that the information is directly in front of her eyes. This obviates the need for large banks of information to be scanned with the corresponding lack of attention to what is happening outside, and makes the pilot's job easier. Less important information is usually presented on a smaller number of dials and gauges in the cockpit to avoid cluttering the head-up display, and these can be monitored less often, during times of low stress.

2.6.2 Sound output

Another mode of output that we should consider is that of auditory signals. Often designed to be used in conjunction with screen displays, auditory outputs are poorly understood: we do not yet know how to utilize sound in a sensible way to achieve maximum effect and information transference. We have discussed speech previously, but other sounds such as beeps, bongs, clanks, whistles and whirrs are all used to varying effect. As well as conveying system output, sounds offer an important level of feedback in interactive systems. Keyboards can be set to emit a click each time a key is pressed, and this appears to speed up interactive performance. Telephone keypads often sound different tones when the keys are pressed; a noise occurring signifies that the key has been successfully pressed, whilst the actual tone provides some information about the particular key that was pressed. The advantage of auditory feedback is evident when we consider a simple device such as a doorbell. If we press it and hear nothing, we are left undecided. Should we press it again, in case we did not do it right the first time, or did it ring but we did not hear it? And if we press it again but it actually did ring, will the people in the house think we are very rude, ringing insistently? We feel awkward and a little stressed. If we were using a computer system instead of a doorbell and were faced with a similar problem, we would not enjoy the interaction and would not perform as well. Yet it is a simple problem that could be easily rectified by a better initial design, using sound. Chapter 10 will discuss the use of the auditory channel in more detail.

2.6.3 Touch, feel and smell

Our other senses are used less in normal computer applications, but you may have played computer games where the joystick or artificial steering wheel vibrated, perhaps when a car was about to go off the track. In some VR applications, such as the use in medical domains to 'practice' surgical procedures, the *feel* of an instrument moving through different tissue types is very important. The devices used to emulate these procedures have *force feedback*, giving different amounts of resistance depending on the state of the virtual operation. These various forms of force, resistance and texture that influence our physical senses are called *haptic* devices.

Haptic devices are not limited to virtual environments, but are used in specialist interfaces in the real world too. Electronic braille displays either have pins that rise or fall to give different patterns, or may involve small vibration pins. Force feedback has been used in the design of in-car controls.

In fact, the car gives a very good example of the power of tactile feedback. If you drive over a small bump in the road the car is sent slightly off course; however, the chances are that you will correct yourself before you are consciously aware of the bump. Within your body you have reactions that push back slightly against pressure to keep your limbs where you 'want' them, or move your limbs out of the way when you brush against something unexpected. These responses occur in your lower brain and are very fast, not involving any conscious effort. So, haptic devices can access very fast responses, but these responses are not fully controlled. This can be used effectively in design, but of course also with caution.

Texture is more difficult as it depends on small changes between neighboring points on the skin. Also, most of our senses notice change rather than fixed stimuli, so we usually feel textures when we move our fingers over a surface, not just when resting on it. Technology for this is just beginning to become available

There is evidence that smell is one of the strongest cues to memory. Various historical recreations such as the Jorvik Centre in York, England, use smells to create a feeling of immersion in their static displays of past life. Some arcade games also generate smells, for example, burning rubber as your racing car skids on the track. These examples both use a fixed smell in a particular location. There have been several attempts to produce devices to allow smells to be recreated dynamically in response to games or even internet sites. The technical difficulty is that our noses do not have a small set of basic smells that are mixed (like salt/sweet/sour/bitter/savoury on our tongue), but instead there are thousands of different types of receptor responding to different chemicals in the air. The general pattern of devices to generate smells is to have a large repertoire of tiny scent-containing capsules that are released in varying amounts on demand – rather like a printer cartridge with hundreds of ink colors! So far there appears to be no mass market for these devices, but they may eventually develop from niche markets.

Smell is a complex multi-dimensional sense and has a peculiar ability to trigger memory, but cannot be changed rapidly. These qualities may prove valuable in areas where a general sense of location and awareness is desirable. For example, a project at the Massachusetts Institute of Technology explored the use of a small battery of scent generators which may be particularly valuable for *ambient displays* and background awareness [198, 161].

Design Focus

Feeling the road

In the BMW 7 Series you will find a single haptic feedback control for many of the functions that would normally have dedicated controls. It uses technology developed by Immersion Corporation who are also behind the force feedback found in many medical and entertainment haptic devices. The iDrive control slides backwards and forwards and rotates to give access to various menus and lists of options. The haptic feedback allows the user to feel 'clicks' appropriate to the number of items in the various menu lists.

See: www.immersion.com/ and www.bmw.com/ Picture courtesy of BMW AG

2.6.4 Physical controls

Look at Figure 2.13. In it you can see the controls for a microwave, a washing machine and a personal MiniDisc player. See how they each use very different physical devices: the microwave has a flat plastic sheet with soft buttons, the washing machine large switches and knobs, and the MiniDisc has small buttons and an interesting multi-function end.

A desktop computer system has to serve many functions and so has generic keys and controls that can be used for a variety of purposes. In contrast, these dedicated control panels have been designed for a particular device and for a single use. This is why they differ so much.

Looking first at the microwave, it has a flat plastic control panel. The buttons on the panel are pressed and 'give' slightly. The choice of the smooth panel is probably partly for visual design – it looks streamlined! However, there are also good practical reasons. The microwave is used in the kitchen whilst cooking, with hands that may be greasy or have food on them. The smooth controls have no gaps where food can accumulate and clog buttons, so it can easily be kept clean and hygienic.

When using the washing machine you are handling dirty clothes, which may be grubby, but not to the same extent, so the smooth easy-clean panel is less important (although some washing machines do have smooth panels). It has several major

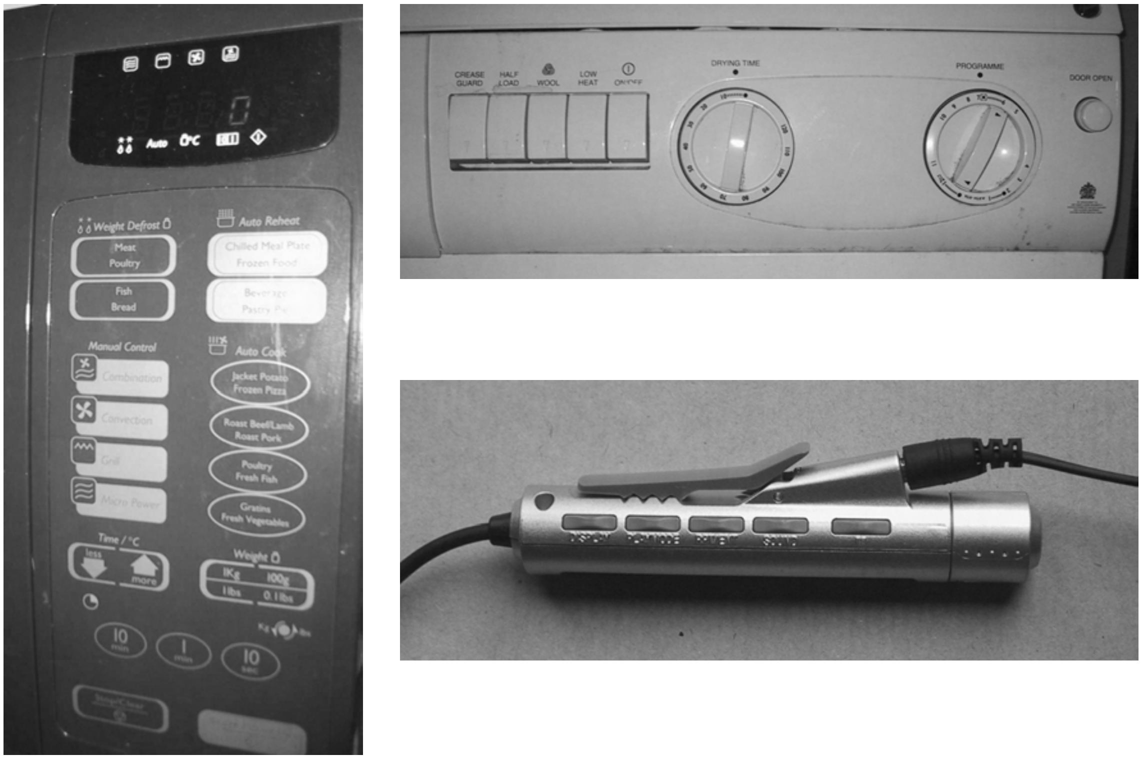


Figure 2.13 Physical controls on microwave, washing machine and MiniDisc. Source: Photograph bottom right by Alan Dix with permission from Sony (UK)

settings and the large buttons act both as control and display. Also the dials for dryer timer and the washing program act both as a means to set the desired time or program and to display the current state whilst the wash is in progress.

Finally, the MiniDisc controller needs to be small and unobtrusive. It has tiny buttons, but the end control is most interesting. It twists from side to side and also can be pulled and twisted. This means the same control can be used for two different purposes. This form of multi-function control is common in small devices.

We discussed the immediacy of haptic feedback and these lessons are also important at the level of creating physical devices; do keys, dials, etc., feel as if they have been pressed or turned? Getting the right level of resistance can make the device work naturally, give you feedback that you have done something, or let you know that you are controlling something. Where for some reason this is not possible, something has to be done to prevent the user getting confused, perhaps pressing buttons twice; for example, the smooth control panel of the microwave in Figure 2.13 offers no tactile feedback, but beeps for each keypress. We will discuss these design issues further when we look at user experience in Chapter 3 (Section 3.9).

Whereas texture is difficult to generate, it is easy to build into materials. This can make a difference to the ease of use of a device. For example, a touchpad is smooth, but a keyboard nipple is usually rubbery. If they were the other way round it would be hard to drag your finger across the touchpad or to operate the nipple without slipping. Texture can also be used to disambiguate. For example, most keyboards have a small raised dot on the 'home' keys for touch typists and some calculators and phones do the same on the '5' key. This is especially useful in applications when the eyes are elsewhere.

2.6.5 Environment and bio-sensing

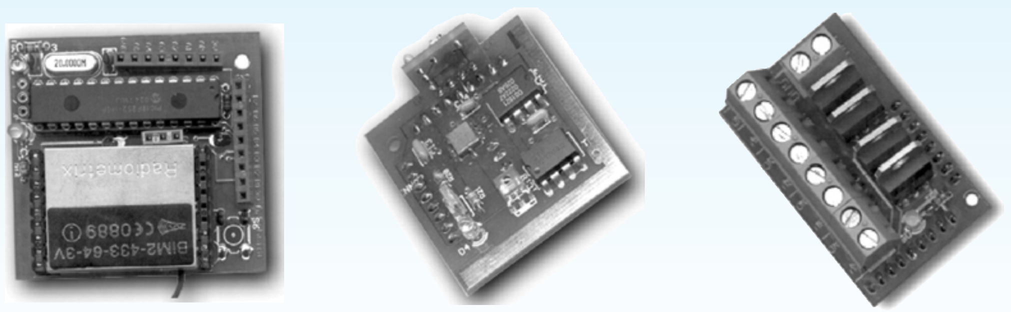
In a public washroom there are often no controls for the wash basins, you simply put your hands underneath and (hope that) the water flows. Similarly when you open the door of a car, the courtesy light turns on. The washbasin is controlled by a small infrared sensor that is triggered when your hands are in the basin (although it is

DESIGN FOCUS

Smart-Its – making using sensors easy

Building systems with physical sensors is no easy task. You need a soldering iron, plenty of experience in electronics, and even more patience. Although some issues are unique to each sensor or project, many of the basic building blocks are similar – connecting simple microprocessors to memory and networks, connecting various standard sensors such as temperature, tilt, etc.

The Smart-Its project has made this job easier by creating a collection of components and an architecture for adding new sensors. There are a number of basic Smart-It boards – the photo on the left shows a microprocessor with wireless connectivity. Onto these boards are plugged a variety of modules – in the center is a sensor board including temperature and light, and on the right is a power controller.



See: www.smart-its.org/ Source: Courtesy of Hans Gellersen sometimes hard to find the 'sweet spot' where this happens!). The courtesy lights are triggered by a small switch in the car door.

Although we are not always conscious of them, there are many sensors in our environment – controlling automatic doors, energy saving lights, etc. and devices monitoring our behavior such as security tags in shops. The vision of ubiquitous computing (see Chapters 4 and 20) suggests that our world will be filled with such devices. Certainly the gap between science fiction and day-to-day life is narrow; for example, in the film *Minority Report* (20th Century Fox) iris scanners identify each passer-by to feed them dedicated advertisements, but you can buy just such an iris scanner as a security add-on for your home computer.

There are many different sensors available to measure virtually anything: temperature, movement (ultrasound, infrared, etc.), location (GPS, global positioning, in mobile devices), weight (pressure sensors). In addition audio and video information can be analyzed to identify individuals and to detect what they are doing. This all sounds big brother like, but is also used in ordinary applications, such as the washbasin.

Sensors can also be used to capture physiological signs such as body temperature, unconscious reactions such as blink rate, or unconscious aspects of activities such as typing rate, vocabulary shifts (e.g. modal verbs). For example, in a speech-based game, Tsukahara and Ward use gaps in speech and prosody (patterns of rhythm, pitch and loudness in speech) to infer the user's emotional state and thus the nature of acceptable responses [350] and Allanson discusses a variety of physiological sensors to create 'electrophysiological interactive computer systems' [12].

2.7 PAPER: PRINTING AND SCANNING

Some years ago, a recurrent theme of information technology was the *paperless office*. In the paperless office, documents would be produced, dispatched, read and filed online. The only time electronic information would be committed to paper would be when it went out of the office to ordinary customers, or to other firms who were laggards in this technological race. This vision was fuelled by rocketing property prices, and the realization that the floor space for a wastepaper basket could cost thousands in rent each year. Some years on, many traditional paper files are now online, but the desire for the completely paperless office has faded. Offices still have wastepaper baskets, and extra floor space is needed for the special computer tables to house 14-inch color monitors.

In this section, we will look at some of the available technology that exists to get information to and from paper. We will look first at printing, the basic technology, and issues raised by it. We will then go on to discuss the movement from paper back into electronic media. Although the paperless office is no longer seen as the goal, the less-paper office is perhaps closer, now that the technologies for moving between media are better.

2.7.1 Printing

If anything, computer systems have made it easier to produce paper documents. It is so easy to run off many copies of a letter (or book), in order to get it looking ‘just right’. Older printers had a fixed set of characters available on a printhead. These varied from the traditional line printer to golf-ball and daisy-wheel printers. To change a typeface or the size of type meant changing the printhead, and was an awkward, and frequently messy, job, but for many years the daisy-wheel printer was the only means of producing high-quality output at an affordable price. However, the drop in the price of laser printers coupled with the availability of other cheap high-quality printers means that daisy-wheels are fast becoming a rarity.

All of the popular printing technologies, like screens, build the image on the paper as a series of dots. This enables, in theory, any character set or graphic to be printed,

Common types of dot-based printers

Dot-matrix printers
These use an inked ribbon, like a typewriter, but instead of a single character-shaped head striking the paper, a line of *pins* is used, each of which can strike the ribbon and hence dot the paper. Horizontal resolution can be varied by altering the speed of the head across the paper, and vertical resolution can be improved by sending the head twice across the paper at a slightly different position. So, dot-matrix printers can produce fast draft-quality output or slower ‘letter’-quality output. They are cheap to run, but could not compete with the quality of jet and laser printers for general office and home printing. They are now only used for bulk printing, or where carbon paper is required for payslips, check printing, etc.)

Ink-jet and bubble-jet printers
These operate by sending tiny blobs of ink from the printhead to the paper. The ink is squirted at pressure from an ink-jet, whereas bubble-jets use heat to create a bubble. Both are quite quiet in operation. The ink from the bubble-jet (being a bubble rather than a droplet) dries more quickly than the ink-jet and so is less likely to smear. Both approach laser quality, but the bubble-jet dots tend to be more accurately positioned and of a less broken shape.

Laser printer
This uses similar technology to a photocopier: ‘dots’ of electrostatic charge are deposited on a drum, which then picks up toner (black powder). This is then rolled onto the paper and cured by heat. The curing is why laser printed documents come out warm, and the electrostatic charge is why they smell of ozone! In addition, some toner can be highly toxic if inhaled, but this is more a problem for full-time maintenance workers than end-users changing the occasional toner cartridge.

Laser printers give nearly typeset-quality output, with top-end printers used by desktop publishing firms. Indeed, many books are nowadays produced using laser printers. The authors of this book have produced camera-ready copy for other books on 300 and 600 dpi laser printers, although this book required higher quality and the first edition was typeset at 1200 dpi onto special bromide paper.

limited only by the resolution of the dots. This resolution is measured in *dots per inch* (dpi). Imagine a sheet of graph paper, and building up an image by putting dots at the intersection of each line. The number of lines per inch in each direction is the resolution in dpi. For some mechanical printers this is slightly confused: the dots printed may be bigger than the gaps, neighboring printheads may not be able to print simultaneously and may be offset relative to one another (a diamond-shaped rather than rectangular grid). These differences do not make too much difference to the user, but mean that, given two printers at the same nominal resolution, the output of one looks better than that of the other, because it has managed the physical constraints better.

The most common types of dot-based printers are dot-matrix printers, ink-jet printers and laser printers. These are listed roughly in order of increasing resolution and quality, where dot-matrix printers typically have a resolution of 80–120 dpi rising to about 300–600 dpi for ink-jet printers and 600–2400 dpi for laser printers. By varying the quantity of ink and quality of paper, ink-jet printers can be used to print photo-quality prints from digital photographs.

Printing in the workplace

Although ink-jet and laser printers have the lion's share of the office and home printer market, there are many more specialist applications that require different technology.

Most shop tills use dot-matrix printing where the arrangement is often very clever, with one printhead serving several purposes. The till will usually print one till roll which stays within the machine, recording all transactions for audit purposes. An identical receipt is printed for the customer. In addition, many will print onto the customer's own check or produce a credit card slip for the customer to sign. Sometimes the multiple copies are produced using two or more layers of paper where the top layer receives the ink and the lower layers use pressure-sensitive paper – not possible using ink-jet or laser technology. Alternatively, a single printhead may move back and forth over several small paper rolls within the same machine, as well as moving over the slot for the customer's own check.

As any printer owner will tell you, office printers are troublesome, especially as they age. Different printing technology is therefore needed in harsh environments or where a low level of supervision is required. Thermal printers use special heat-sensitive paper that changes color when heated. The printhead simply heats the paper where it wants a dot. Often only one line of dots is produced per pass, in contrast to dot-matrix and ink-jet printers, which have several pins or jets in parallel. The image is then produced using several passes per line, achieving a resolution similar to a dot-matrix. Thermal paper is relatively expensive and not particularly nice to look at, but thermal printers are mechanically simple and require little maintenance (no ink or toner splashing about). Thermal printers are used in niche applications, for example industrial equipment, some portable printers, and fax machines (although many now use plain paper).

As well as resolution, printers vary in speed and cost. Typically, office-quality ink-jet or laser printers produce between four and eight pages per minute. Dot-matrix printers are more often rated in *characters per second* (cps), and typical speeds may be 200 cps for draft and 50 cps for letter-quality print. In practice, this means no more than a page or so per minute. These are maximum speeds for simple text, and printers may operate much more slowly for graphics.

Color ink-jet printers are substantially cheaper than even monochrome laser printers. However, the recurrent costs of consumables may easily dominate this initial cost. Both jet and laser printers have special-purpose parts (print cartridges, toner, print drums), which need to be replaced every few thousand sheets; and they must also use high-grade paper. It may be more difficult to find suitable grades of recycled paper for laser printers.

2.7.2 Fonts and page description languages

Some printers can act in a mode whereby any characters sent to them (encoded in ASCII, see Section 2.8.5) are printed, typewriter style, in a single font. Another case, simple in theory, is when you have a bitmap picture and want to print it. The dots to print are sent to the printer, and no further interpretation is needed. However, in practice, it is rarely so simple.

Many printed documents are far more complex – they incorporate text in many different fonts and many sizes, often italicized, emboldened and underlined. Within the text you will find line drawings, digitized photographs and pictures generated from ‘paint’ packages, including the ubiquitous ‘clip art’. Sometimes the computer does all the work, converting the page image into a bitmap of the right size to be sent to the printer. Alternatively, a description of the page may be sent to the printer. At the simplest level, this will include commands to set the print position on the page, and change the font size.

More sophisticated printers can accept a *page description language*, the most common of which is PostScript. This is a form of programming language for printing. It includes some standard programming constructs, but also some special ones: paths for drawing lines and curves, sophisticated character and font handling and scaled bitmaps. The idea is that the description of a page is far smaller than the associated bitmap, reducing the time taken to send the page to the printer. A bitmap version of an A4 laser printer page at 300 dpi takes 8 Mbytes; to send this down a standard serial printer cable would take 10 minutes! However, a computer in the printer has to interpret the PostScript program to print the page; this is typically faster than 10 minutes, but is still the limiting factor for many print jobs.

Text is printed in a font with a particular size and shape. The size of a font is measured in points (pt). The point is a printer’s measure and is about 1/72 of an inch. The *point size* of the font is related to its height: a 12 point font has about six lines per inch. The shape of a font is determined by its *font name*, for example Times Roman, Courier or Helvetica. Times Roman font is similar to the type of many newspapers, such as *The Times*, whereas Courier has a typewritten shape.

```
Courier is a fixed-pitch font
```
Times Roman is a variable-pitch serif font

Minion is also a variable-pitch serif font

Gill Sans is a variable-pitch sans-serif font

A mathematics font: αβξ±π∈∀∞⊥≠ℵ∂√∃

Figure 2.14 Examples of different fonts

Some fonts, such as Courier, are *fixed pitch*, that is each character has the same width. The alternative is a variable-pitched font, such as Times Roman or Gill Sans, where some characters, such as the ‘m’, are wider than others, such as the ‘i’. Another characteristic of fonts is whether they are *serif* or *sans-serif*. A serif font has fine, short cross-lines at the ends of the strokes, imitating those found on cut stone lettering. A sans-serif font has square-ended strokes. In addition, there are special fonts looking like Gothic lettering or cursive script, and fonts of Greek letters and special mathematical symbols.

This book is set in 10 point Minion font using PostScript. Minion is a variable-pitched serif font. Figure 2.14 shows examples of different fonts.

Design Focus

Readability of text

There is a substantial body of knowledge about the readability of text, both on screen and on paper. An MSc student visited a local software company and, on being shown some of their systems, remarked on the fact that they were using upper case throughout their displays. At that stage she had only completed part of an HCI course but she had read Chapter 1 of this book and already knew that WORDS WRITTEN IN BLOCK CAPITALS take longer to read than those in lower case. Recall that this is largely because of the clues given by word shapes and is the principle behind ‘look and say’ methods of teaching children to read. The company immediately recognized the value of the advice and she instantly rose in their esteem!

However, as with many interface design guidelines there are caveats. Although lower-case words are easier to read, individual letters and nonsense words are clearer in upper case. For example, one writes flight numbers as ‘BA793’ rather than ‘ba793’. This is particularly important when naming keys to press (for example, ‘Press Q to quit’) as keyboards have upper-case legends.

Font shapes can also make a difference; for printed text, serif fonts make it easier to run one’s eye along a line of text. However, they usually reproduce less well on screen where the resolution is poorer.

2.7.3 Screen and page

A common requirement of word processors and desktop publishing software is that *what you see is what you get* (see also Chapters 4 and 17), which is often called by its acronym *WYSIWYG* (pronounced whizz-ee-wig). This means that the appearance of the document on the screen should be the same as its eventual appearance on the printed page. In so far as this means that, for example, centered text is displayed centered on the screen, this is reasonable. However, this should not cloud the fact that screen and paper are very different media.

A typical screen resolution is about 72 dpi compared with a laser printer at over 600 dpi. Some packages can show magnified versions of the document in order to help in this. Most screens use an additive color model using red, green and blue light, whereas printers use a subtractive color model with cyan, magenta, yellow and black inks, so conversions have to be made. In addition, the sizes and aspect ratios are very different. An A4 page is about 11 inches tall by 8 wide (297 × 210 mm), whereas a screen is often of similar dimensions, but wider than it is tall.

These differences cause problems when designing software. Should you try to make the screen image as close to the paper as possible, or should you try to make the best of each? One approach to this would be to print only what could be displayed, but that would waste the extra resolution of the printer. On the other hand, one can try to make the screen as much like paper as possible, which is the intention behind the standard use of black text on a white background, rotatable A4 displays, and tablet PCs. This is a laudable aim, but cannot get rid of all the problems.

A particular problem lies with fonts. Imagine we have a line of ‘m’s, each having a width of 0.15 inch (4 mm). If we print them on a 72 dpi screen, then we can make the screen character 10 or 11 dots wide, in which case the screen version will be narrower or wider than the printed version. Alternatively, we can print the screen version as near as possible to where the printed characters would lie, in which case the ‘m’s on the screen would have different spaces between them: ‘mm mm mm mm m’. The latter looks horrible on the screen, so most software chooses the former approach. This means that text that aligns on screen may not do so on printing. Some systems use a uniform representation for screen and printer, using the same font descriptions and even, in the case of the Next operating system, PostScript for screen display as well as printer output (also PDF with MacOS X). However, this simply exports the problem from the application program to the operating system.

The differences between screen and printer mean that different forms of graphic design are needed for each. For example, headings and changes in emphasis are made using font style and size on paper, but using color, brightness and line boxes on screen. This is not usually a problem for the display of the user’s own documents as the aim is to give the user as good an impression of the printed page as possible, given the limitations. However, if one is designing parallel paper and screen forms, then one has to trade off consistency between the two representations with clarity in each.

An overall similar layout, but with different forms of presentation for details, may be appropriate.

2.7.4 Scanners and optical character recognition

Printers take electronic documents and put them on paper – *scanners* reverse this process. They start by turning the image into a bitmap, but with the aid of *optical character recognition* can convert the page right back into text. The image to be converted may be printed, but may also be a photograph or hand-drawn picture.

There are two main kinds of scanner: flat-bed and hand-held. With a flat-bed scanner, the page is placed on a flat glass plate and the whole page is converted into a bitmap. A variant of the flat-bed is where sheets to be scanned are pulled through the machine, common in multi-function devices (printer/fax/copier). Many flat-bed scanners allow a small pile of sheets to be placed in a feed tray so that they can all be scanned without user intervention. Hand-held scanners are pulled over the image by hand. As the head passes over an area it is read in, yielding a bitmap strip. A roller at the ends ensures that the scanner knows how fast it is being pulled and thus how big the image is. The scanner is typically only 3 or 4 inches (80 or 100 mm) wide and may even be the size of a large pen (mainly used for scanning individual lines of text). This means at least two or three strips must be 'glued' together by software to make a whole page image, quite a difficult process as the strips will overlap and may not be completely parallel to one another, as well as suffering from problems of different brightness and contrast. However, for desktop publishing small images such as photographs are quite common, and as long as one direction is less than the width of the scanner, they can be read in one pass.

Scanners work by shining a beam of light at the page and then recording the intensity and color of the reflection. Like photocopiers, the color of the light that is shone means that some colors may appear darker than others on a monochrome scanner. For example, if the light is pure red, then a red image will reflect the light completely and thus not appear on the scanned image.

Like printers, scanners differ in resolution, commonly between 600 and 2400 dpi, and like printers the quoted resolution needs careful interpretation. Many have a lower resolution scanhead but digitally interpolate additional pixels – the same is true for some digital cameras. Monochrome scanners are typically only found in multi-function devices, but color scanners usually have monochrome modes for black and white or grayscale copying. Scanners will usually return up to 256 levels of gray or RGB (red, green, blue) color. If a pure monochrome image is required (for instance, from a printed page), then it can *threshold* the grayscale image; that is, turn all pixels darker than some particular value black, and the rest white.

Scanners are used extensively in *desktop publishing* (*DTP*) for reading in hand-drawn pictures and photographs. This means that cut and paste can be performed electronically rather than with real glue. In addition, the images can be rotated, scaled and otherwise transformed, using a variety of image manipulation software tools. Such tools are becoming increasingly powerful, allowing complex image transformations to be easily achieved; these range from color correction, through the merging of multiple images to the application of edge-detection and special effects filters. The use of multiple layers allows photomontage effects that would be impossible with traditional photographic or paper techniques. Even where a scanned image is simply going to be printed back out as part of a larger publication, some processing typically has to be performed to match the scanned colors with those produced during printing. For film photographs there are also special film scanners that can scan photographic negatives or color slides. Of course, if the photographs are digital no scanning is necessary.

Another application area is in document storage and retrieval systems, where paper documents are scanned and stored on computer rather than (or sometimes as well as) in a filing cabinet. The costs of maintaining paper records are enormous, and electronic storage can be cheaper, more reliable and more flexible. Storing a bitmap image is neither most useful (in terms of access methods), nor space efficient (as we will see later), so scanning may be combined with optical character recognition to obtain the text rather than the page image of the document.

Optical character recognition (OCR) is the process whereby the computer can 'read' the characters on the page. It is only comparatively recently that print could be reliably read, since the wide variety of typefaces and print sizes makes this more difficult than one would imagine – it is *not* simply a matter of matching a character shape to the image on the page. In fact, OCR is rather a misnomer nowadays as, although the document is optically scanned, the OCR software itself operates on the bitmap image. Current software can recognize 'unseen' fonts and can even produce output in word-processing formats, preserving super- and subscripts, centering, italics and so on.

Another important area is electronic publishing for multimedia and the world wide web. Whereas in desktop publishing the scanned image usually ends up (after editing) back on paper, in electronic publishing the scanned image is destined to be viewed on screen. These images may be used simply as digital photographs or may be made active, whereby clicking on some portion of the image causes pertinent information to be displayed (see Chapter 3 for more on the *point-and-click* style of interaction). One big problem when using electronic images is the plethora of formats for storing graphics (see Section 2.8.5). Another problem is the fact that different computers can display different numbers of colors and that the appearance of the same image on different monitors can be very different.

The importance of electronic publishing and also the ease of electronically manipulating images for printing have made the *digital camera* increasingly popular. Rather than capturing an image on film, a digital camera has a small light-sensitive chip that can directly record an image into memory.

Paper-based interaction

Paper is principally seen as an output medium. You type in some text, format it, print it and read it. The idea of the paperless office was to remove the paper from the write–read loop entirely, but it didn't fundamentally challenge its place in the cycle as an output medium. However, this view of paper as output has changed as OCR technology has improved and scanners become commonplace.

Workers at Xerox Palo Alto Research Center (also known as Xerox PARC) capitalized on this by using paper as a medium of interaction with computer systems [195]. A special identifying mark is printed onto forms and similar output. The printed forms may have check boxes or areas for writing numbers or (in block capitals!) words. The form can then be scanned back in. The system reads the identifying mark and thereby knows what sort of paper form it is dealing with. It doesn't have to use OCR on the printed text of the form as it printed it, but can detect the check boxes that have been filled in and even recognize the text that has been written. The identifying mark the researchers used is composed of backward and forward slashes, '\' and '/', and is called a *glyph*. An alternative would have been to use bar codes, but the slashes were found to fax and scan more reliably. The research version of this system was known as XAX, but it is now marketed as Xerox PaperWorks.

One application of this technology is mail order catalogs. The order form is printed with a glyph. When completed, forms can simply be collected into bundles and scanned in batches, generating orders automatically. If the customer faxes an order the fax-receiving software recognizes the glyph and the order is processed without ever being handled at the company end. Such a *paper user interface* may involve no screens or keyboards whatsoever.

Some types of paper now have identifying marks micro-printed like a form of textured watermark. This can be used both to identify the piece of paper (as the glyph does), and to identify the location on the paper. If this book were printed on such paper it would be possible to point at a word or diagram with a special pen-like device and have it work out what page you are on and where you are pointing and thus take you to appropriate web materials . . . perhaps the fourth edition . . .

It is paradoxical that Xerox PARC, where much of the driving work behind current 'mouse and window' computer interfaces began, has also developed this totally non-screen and non-mouse paradigm. However, the common principle behind each is the novel and appropriate use of different media for graceful interaction.

Worked exercise

What input and output devices would you use for the following systems? For each, compare and contrast alternatives, and if appropriate indicate why the conventional keyboard, mouse and CRT screen may be less suitable.

(a) portable word processor
(b) tourist information system
(c) tractor-mounted crop-spraying controller

(d) air traffic control system
(e) worldwide personal communications system
(f) digital cartographic system.

Answer In the later exercise on basic architecture (see Section 2.8.6), we focus on 'typical' systems, whereas here the emphasis is on the diversity of different devices needed for specialized purposes. You can 'collect' devices – watch out for shop tills, bank tellers, taxi meters, lift buttons, domestic appliances, etc.

(a) Portable word processor
The determining factors are size, weight and battery power. However, remember the purpose: this is a word processor not an address book or even a data entry device.
(i) LCD screen – low-power requirement
(ii) trackball or stylus for pointing
(iii) real keyboard – you can't word process without a reasonable keyboard and stylus handwriting recognition is not good enough
(iv) small, low-power bubble-jet printer – although not always necessary, this makes the package stand alone. It is probably not so necessary that the printer has a large battery capacity as printing can probably wait until a power point is found.

(b) Tourist information system
This is likely to be in a public place. Most users will only visit the system once, so the information and mode of interaction must be immediately obvious.
(i) touchscreen only – easy and direct interaction for first-time users (see also Chapter 3)
(ii) NO mice or styluses – in a public place they wouldn't stay long!

(c) Tractor-mounted crop-spraying controller
A hostile environment with plenty of mud and chemicals. Requires numerical input for flow rates, etc., but probably no text
(i) touch-sensitive keypad – ordinary keypads would get blocked up
(ii) small dedicated LED display (LCDs often can't be read in sunlight and large screens are fragile)
(iii) again no mice or styluses – they would get lost.

(d) Air traffic control system
The emphasis is on immediately available information and rapid interaction. The controller cannot afford to spend time searching for information; all frequently used information must be readily available.
(i) several specialized displays – including overlays of electronic information on radar
(ii) light pen or stylus – high-precision direct interaction
(iii) keyboard – for occasional text input, but consider making it fold out of the way.

(e) Worldwide personal communications system
Basically a super mobile phone! If it is to be kept on hand all the time it must be very light and pocket sized. However, to be a 'communications' system one would imagine that it should also act as a personal address/telephone book, etc.

(i) standard telephone keypad – the most frequent use
(ii) small dedicated LCD display – low power, specialized functions
(iii) possibly stylus for interaction – it allows relatively rich interaction with the address book software, but little space
(iv) a ‘docking’ facility – the system itself will be too small for a full-sized keyboard(!), but you won’t want to enter in all your addresses and telephone numbers by stylus!

(f) Digital cartographic system
This calls for very high-precision input and output facilities. It is similar to CAD in terms of the screen facilities and printing, but in addition will require specialized data capture.
(i) large high-resolution color VDU (20 inch or bigger) – these tend to be enormously big (from back to front). LCD screens, although promising far thinner displays in the long term, cannot at present be made large enough
(ii) digitizing tablet – for tracing data on existing paper maps. It could also double up as a pointing device for some interaction
(iii) possibly thumbwheels – for detailed pointing and positioning tasks
(iv) large-format printer – indeed very large: an A2 or A1 plotter at minimum.

2.8 MEMORY

Like human memory, we can think of the computer’s memory as operating at different levels, with those that have the faster access typically having less capacity. By analogy with the human memory, we can group these into short-term and long-term memories (STM and LTM), but the analogy is rather weak – the capacity of the computer’s STM is a lot more than seven items! The different levels of computer memory are more commonly called primary and secondary storage.

The details of computer memory are not in themselves of direct interest to the user interface designer. However, the limitations in capacity and access methods are important constraints on the sort of interface that can be designed. After some fairly basic information, we will put the raw memory capacity into perspective with the sort of information which can be stored, as well as again seeing how advances in technology offer more scope for the designer to produce more effective interfaces. In particular, we will see how the capacity of typical memory copes with video images as these are becoming important as part of multimedia applications (see Chapter 21).

2.8.1 RAM and short-term memory (STM)

At the lowest level of computer memory are the registers on the computer chip, but these have little impact on the user except in so far as they affect the general speed of the computer. Most currently active information is held in silicon-chip *random access memory* (*RAM*). Different forms of RAM differ as to their precise access times, power consumption and characteristics. Typical access times are of the order of 10 nanoseconds, that is a hundred-millionth of a second, and information can be accessed at a rate of around 100 Mbytes (million bytes) per second. Typical storage in modern personal computers is between 64 and 256 Mbytes.

Most RAM is *volatile*, that is its contents are lost when the power is turned off. However, many computers have small amount of *non-volatile RAM*, which retains its contents, perhaps with the aid of a small battery. This may be used to store setup information in a large computer, but in a pocket organizer will be the whole memory. Non-volatile RAM is more expensive so is only used where necessary, but with many notebook computers using very low-power static RAM, the divide is shrinking. By strict analogy, non-volatile RAM ought to be classed as LTM, but the important thing we want to emphasize is the gulf between STM and LTM in a traditional computer system.

In PDAs the distinctions become more confused as the battery power means that the system is never completely off, so RAM memory effectively lasts for ever. Some also use flash memory, which is a form of silicon memory that sits between fixed content ROM (read-only memory) chips and normal RAM. Flash memory is relatively slow to write, but once written retains its content even with no power whatsoever. These are sometimes called silicon disks on PDAs. Digital cameras typically store photographs in some form of flash media and small flash-based devices are used to plug into a laptop or desktop's USB port to transfer data.

2.8.2 Disks and long-term memory (LTM)

For most computer users the LTM consists of *disks*, possibly with small tapes for *backup*. The existence of backups, and appropriate software to generate and retrieve them, is an important area for user security. However, we will deal mainly with those forms of storage that impact the interactive computer user.

There are two main kinds of technology used in disks: *magnetic disks* and *optical disks*. The most common storage media, floppy disks and hard (or fixed) disks, are coated with magnetic material, like that found on an audio tape, on which the information is stored. Typical capacities of floppy disks lie between 300 kbytes and 1.4 Mbytes, but as they are removable, you can have as many as you have room for on your desk. Hard disks may store from under 40 Mbytes to several gigabytes (Gbytes), that is several thousand million bytes. With disks there are two access times to consider, the time taken to find the right track on the disk, and the time to read the track. The former dominates random reads, and is typically of the order of 10 ms for hard disks. The transfer rate once the track is found is then very high, perhaps several hundred kilobytes per second. Various forms of large removable media are also available, fitting somewhere between floppy disks and removable hard disks, and are especially important for multimedia storage.

Optical disks use laser light to read and (sometimes) write the information on the disk. There are various high capacity specialist optical devices, but the most common is the *CD-ROM*, using the same technology as audio compact discs. CD-ROMs have a capacity of around 650 megabytes, but cannot be written to at all. They are useful for published material such as online reference books, multimedia and software distribution. Recordable CDs are a form of *WORM* device (write-once read-many) and are more flexible in that information can be written, but (as the name suggests) only once at any location – more like a piece of paper than a blackboard. They are obviously very useful for backups and for producing very secure audit information. Finally, there are fully rewritable optical disks, but the rewrite time is typically much slower than the read time, so they are still primarily for archival not dynamic storage. Many CD-ROM reader/writers can also read DVD format, originally developed for storing movies. Optical media are more robust than magnetic disks and so it is easier to use a *jukebox* arrangement, whereby many optical disks can be brought online automatically as required. This can give an online capacity of many hundreds of gigabytes. However, as magnetic disk capacities have grown faster than the fixed standard of CD-ROMs, some massive capacity stores are moving to large disk arrays.

2.8.3 Understanding speed and capacity

So what effect do the various capacities and speeds have on the user? Thinking of our typical personal computer system, we can summarize some typical capacities as in Table 2.1.

We think first of documents. This book is about 320,000 words, or about 2 Mbytes, so it would hardly make a dent in 256 Mbytes of RAM. (This size – 2 Mbytes – is unformatted and without illustrations; the actual size of the full data files is an order of magnitude bigger, but still well within the capacity of main memory.) To take a more popular work, the Bible would use about 4.5 Mbytes. This would still consume only 2% of main memory, and disappear on a hard disk. However, it might look tight on a smaller PDA. This makes the memory look not too bad, so long as you do not intend to put your entire library online. However, many word processors come with a dictionary and thesaurus, and there is no standard way to use the same one with several products. Together with help files and the program itself, it is not

Table 2.1 Typical capacities of different storage media

	STM small/fast	LTM large/slower
Media:	RAM	Hard disk
Capacity:	256 Mbytes	100 Gbytes
Access time:	10 ns	7 ms
Transfer rate:	100 Mbyte/s	30 Mbyte/s

unusual to find each application consuming tens or even hundreds of megabytes of disk space – it is not difficult to fill a few gigabytes of disk at all!

Similarly, although 256 Mbytes of RAM are enough to hold most (but not all) single programs, windowed systems will run several applications simultaneously, soon using up many megabytes. Operating systems handle this by *paging* unused bits of programs out of RAM onto disk, or even *swapping* the entire program onto disk. This makes little difference to the logical functioning of the program, but has a significant effect on interaction. If you select a window, and the relevant application happens to be currently swapped out onto the disk, it has to be swapped back in. The delay this causes can be considerable, and is both noticeable and annoying on many systems.

Technological change and storage capacity

Most of the changes in this book since the first and second editions have been additions where new developments have come along. However, this portion has had to be scrutinized line by line as the storage capacities of high-end machines when this book was first published in 1993 looked ridiculous as we revised it in 1997 and then again in 2003. One of our aims in this chapter was to give readers a concrete feel for the capacities and computational possibilities in standard computers. However, the pace of advances in this area means that it becomes out of date almost as fast as it is written! This is also a problem for design; it is easy to build a system that is sensible given a particular level of technology, but becomes meaningless later. The solution is either to issue ever more frequent updates and new versions, or to exercise a bit of foresight . . .

The delays due to swapping are a symptom of the *von Neumann bottleneck* between disk and main memory. There is plenty of information in the memory, but it is not where it is wanted, in the machine's RAM. The path between them is limited by the transfer rate of the disk and is too slow. Swapping due to the operating system may be difficult to avoid, but for an interactive system designer some of these problems can be avoided by thinking carefully about where information is stored and when it is transferred. For example, the program can be *lazy* about information transfer. Imagine the user wants to look at a document. Rather than reading in the whole thing before letting the user continue, just enough is read in for the first page to be displayed, and the rest is read during idle moments.

Returning to documents, if they are scanned as bitmaps (and not read using OCR), then the capacity of our system looks even less impressive. Say an 11 × 8 inch (297 × 210 mm) page is scanned with an 8 bit grayscale (256 levels) setting at 1200 dpi. The image contains about one billion bits, that is about 128 Mbyte. So, our 100 Gbyte disk could store 800 pages – just OK for this book, but not for the Bible.

If we turn to video, things are even worse. Imagine we want to store moving video using 12 bits for each pixel (4 bits for each primary color giving 16 levels of brightness), each frame is 512 × 512 pixels, and we store at 25 frames per second.

This is by no means a high-quality image, but each frame requires approximately 400 kbytes giving 10 Mbytes per second. Our disk will manage about three hours of video – one good movie. Lowering our sights to still photographs, good digital cameras usually take 2 to 4 mega pixels at 24 bit color; that is 10 Mbytes of raw uncompressed image – you'd not get all your holiday snaps into main memory!

2.8.4 Compression

In fact, things are not quite so bad, since *compression* techniques can be used to reduce the amount of storage required for text, bitmaps and video. All of these things are highly redundant. Consider text for a moment. In English, we know that if we use the letter 'q' then 'u' is almost bound to follow. At the level of words, some words like 'the' and 'and' appear frequently in text in general, and for any particular work one can find other common terms (this book mentions 'user' and 'computer' rather frequently). Similarly, in a bitmap, if one bit is white, there is a good chance the next will be as well. Compression algorithms take advantage of this redundancy. For example, *Huffman encoding* gives short codes to frequent words [182], and *run-length encoding* represents long runs of the same value by length value pairs. Text can easily be reduced by a factor of five and bitmaps often compress to 1% of their original size.

For video, in addition to compressing each frame, we can take advantage of the fact that successive frames are often similar. We can compute the *difference* between successive frames and then store only this – compressed, of course. More sophisticated algorithms detect when the camera pans and use this information also. These differencing methods fail when the scene changes, and so the process periodically has to restart and send a new, complete (but compressed) image. For storage purposes this is not a problem, but when used for transmission over telephone lines or networks it can mean glitches in the video as the system catches up.

With these reductions it is certainly possible to store low-quality video at 64 kbyte/s; that is, we can store five hours of highly compressed video on our 1 Gbyte hard disk. However, it still makes the humble video cassette look very good value.

Probably the leading edge of video still and photographic compression is *fractal compression*. Fractals have been popularized by the images of the *Mandelbrot set* (that swirling pattern of computer-generated colors seen on many T-shirts and posters). Fractals refer to any image that contains parts which, when suitably scaled, are similar to the whole. If we look at an image, it is possible to find parts which are approximately self-similar, and these parts can be stored as a fractal with only a few numeric parameters. Fractal compression is especially good for textured features, which cause problems for other compression techniques. The *decompression* of the image can be performed to any degree of accuracy, from a very rough soft-focus image, to one *more* detailed than the original. The former is very useful as one can produce poor-quality output quickly, and better quality given more time. The latter is rather remarkable – the fractal compression actually fills in details that are not in the original. These details are not accurate, but look convincing!

2.8.5 Storage format and standards

The most common data types stored by interactive programs are text and bitmap images, with increasing use of video and audio, and this subsection looks at the ridiculous range of file storage standards. We will consider database retrieval in the next subsection.

The basic standard for text storage is the *ASCII* (American standard code for information interchange) character codes, which assign to each standard printable character and several control characters an internationally recognized 7 bit code (decimal values 0–127), which can therefore be stored in an 8 bit byte, or be transmitted as 8 bits including parity. Many systems extend the codes to the values 128–255, including line-drawing characters, mathematical symbols and international letters such as 'æ'. There is a 16 bit extension, the UNICODE standard, which has enough room for a much larger range of characters including the Japanese Kanji character set.

As we have already discussed, modern documents consist of more than just characters. The text is in different fonts and includes formatting information such as centering, page headers and footers. On the whole, the storage of formatted text is vendor specific, since virtually every application has its own file format. This is not helped by the fact that many suppliers attempt to keep their file formats secret, or update them frequently to stop others' products being compatible. With the exception of bare ASCII, the most common shared format is *rich text format* (*RTF*), which encodes formatting information including style sheets. However, even where an application will import or export RTF, it may represent a cut-down version of the full document style.

RTF regards the document as formatted text, that is it concentrates on the appearance. Documents can also be regarded as structured objects: this book has chapters containing sections, subsections . . . paragraphs, sentences, words and characters. There are *ISO standards* for document structure and interchange, which in theory could be used for transfer between packages and sites, but these are rarely used in practice. Just as the PostScript language is used to describe the printed page, *SGML* (*standard generalized markup language*) can be used to store structured text in a reasonably extensible way. You can define your own structures (the definition itself in SGML), and produce documents according to them. XML (extensible markup language), a lightweight version of SGML, is now used extensively for web-based applications.

For bitmap storage the range of formats is seemingly unending. The stored image needs to record the size of the image, the number of bits per pixel, possibly a color map, as well as the bits of the image itself. In addition, an icon may have a 'hot-spot' for use as a cursor. If you think of all the ways of encoding these features, or leaving them implicit, and then consider all the combinations of these different encodings, you can see why there are problems. And all this before we have even considered the effects of compression! There is, in fact, a whole software industry producing packages that convert from one format to another.

Given the range of storage standards (or rather lack of standards), there is no easy advice as to which is best, but if you are writing a new word processor and are about to decide how to store the document on disk, think, just for a moment, before defining yet another format.

2.8.6 Methods of access

Standard database access is by special key fields with an associated index. The user has to know the key before the system can find the information. A telephone directory is a good example of this. You can find out someone's telephone number if you know their name (the key), but you cannot find the name given the number. This is evident in the interface of many computer systems. So often, when you contact an organization, they can only help you if you give your customer number, or last order number. The usability of the system is seriously impaired by a shortsighted reliance on a single key and index. In fact, most database systems will allow multiple keys and indices, allowing you to find a record given partial information. So these problems are avoidable with only slight foresight.

There are valid reasons for not indexing on too many items. Adding extra indices adds to the size of the database, so one has to balance ease of use against storage cost. However, with ever-increasing disk sizes, this is not a good excuse for all but extreme examples. Unfortunately, brought up on lectures about algorithmic efficiency, it is easy for computer scientists to be stingy with storage. Another, more valid, reason for restricting the fields you index is privacy and security. For example, telephone companies will typically hold an online index that, given a telephone number, would return the name and address of the subscriber, but to protect the privacy of their customers, this information is not divulged to the general public.

It is often said that dictionaries are only useful for people who can spell. Bad spellers do not know what a word looks like so cannot look it up to find out. Not only in spelling packages, but in general, an application can help the user by matching badly spelt versions of keywords. One example of this is *do what I mean* (*DWIM*) used in several of Xerox PARC's experimental programming environments. If a command name is misspelt the system prompts the user with a close correct name. Menu-based systems make this less of an issue, but one can easily imagine doing the same with, say, file selection. Another important instance of this principle is *Soundex*, a way of indexing words, especially names. Given a key, Soundex finds those words which sound similar. For example, given McCloud, it would find MacCleod. These are all examples of *forgiving systems*, and in general one should aim to accommodate the user's mistakes. Again, there are exceptions to this: you do not want a bank's automated teller machine (ATM) to give money when the PIN number is *almost* correct!

Not all databases allow long passages of text to be stored in records, perhaps setting a maximum length for text strings, or demanding the length be fixed in advance. Where this is the case, the database seriously restricts interface applications where text forms an important part. At the other extreme, *free text retrieval* systems are centered on unformatted, unstructured text. These systems work by keeping an index of every word in every document, and so you can ask 'give me all documents with the words "human" and "computer" in them'. Programs, such as versions of the UNIX 'grep' command, give some of the same facilities by quickly scanning a list of files for a certain word, but are much slower. On the web, free text search is of course the standard way to find things using search engines.

Worked exercise *What is the basic architecture of a computer system?*

Answer In an HCI context, you should be assessing the architecture from the point of view of the user. The material for this question is scattered throughout the chapter. Look too at personal computer magazines, where adverts and articles will give you some idea of typical capabilities . . . and costs. They may also raise some questions: just what is the difference to the user between an 8 ms and a 10 ms disk drive?

The example answer below gives the general style, although more detail would be expected of a full answer. In particular, you need to develop a feel for capacities either as ball-park figures or in terms of typical capabilities (seconds of video, pages of text).

Example
The basic architecture of a computer system consists of the computer itself (with associated memory), input and output devices for user interaction and various forms of hard-copy devices. (Note, the 'computer science' answer regards output to the user and output to a printer as essentially equivalent. This is not an acceptable user-centered view.)

A typical configuration of user input–output devices would be a screen with a keyboard for typing text and a mouse for pointing and positioning. Depending on circumstance, different pointing devices may be used such as a stylus (for more direct interaction) or a touchpad (especially on portable computers).

The computer itself can be considered as composed of some processing element and memory. The memory is itself divided into short-term memory which is lost when the machine is turned off and permanent memory which persists.

2.9 PROCESSING AND NETWORKS

Computers that run interactive programs will process in the order of 100 million instructions per second. It sounds a lot and yet, like memory, it can soon be used up. Indeed, the first program written by one of the authors (some while ago) 'hung' and all attempts to debug it failed. Later calculation showed that the program would have taken more than the known age of the universe to complete! Failures need not be as spectacular as that to render a system unusable. Consider, for example, one drawing system known to the authors. To draw a line you press down the mouse button at one end, drag the mouse and then release the mouse button at the other end of the line – but not too quickly. You have to press down the button and then actually hold your hand steady for a moment, otherwise the line starts half way! For activities involving the user's hand–eye coordination, delays of even a fraction of a second can be disastrous.

Moore's law

Everyone knows that computers just get faster and faster. However, in 1965 Gordon Moore, co-founder of Intel, noticed a regularity. It seemed that the speed of processors, related closely to the number of transistors that could be squashed on a silicon wafer, was doubling every 18 months – exponential growth. One of the authors bought his first 'proper' computer in 1987; it was a blindingly fast 1.47 MHz IBM compatible (Macs were too expensive). By 2002 a system costing the same in real terms would have had a 1.5 GHz processor – 1000 times faster or 2^{10} in 15 years, that is 10×18 months.

There is a similar pattern for computer memory, except that the doubling time for magnetic storage seems to be closer to one year. For example, when the first edition of this book was written one of the authors had a 20 Mbyte hard disk; now, 11 years later, his disk is 30 Gbytes – around 2^{10} times more storage in just 10 years.

The effects of this are dramatic. If you took a young baby today and started recording a full audio video diary of every moment, day and night, of that child's life, by the time she was an old lady her whole life experience would fit into memory the size of a small grain of dust.

For more on Moore's law and life recording see: /e3/online/moores-law/

2.9.1 Effects of finite processor speed

As we can see, speed of processing can seriously affect the user interface. These effects must be taken into account when designing an interactive system. There are two sorts of faults due to processing speed: those when it is too slow, and those when it is too fast!

We saw one example of the former above. This was a *functional fault*, in that the program did the wrong thing. The system is supposed to draw lines from where the mouse button is depressed to where it is released. However, the program gets it wrong – after realizing the button is down, it does not check the position of the mouse fast enough, and so the user may have moved the mouse before the start position is registered. This is a fault at the implementation stage of the system rather than of the design. But to be fair, the programmer may not be given the right sort of information from lower levels of system software.

A second fault due to slow processing is where, in a sense, the program does the right thing, but the feedback is too slow, leading to strange effects at the interface. In order to avoid faults of the first kind, the system *buffers* the user input; that is, it remembers keypresses and mouse buttons and movement. Unfortunately, this leads to problems of its own. One example of this sort of problem is *cursor tracking*, which happens in character-based text editors. The user is trying to move backwards on the same line to correct an error, and so presses the cursor-left key. The cursor moves and when it is over the correct position, the user releases the key. Unfortunately, the system is behind in responding to the user, and so has a few more cursor-left keys to process – the cursor then overshoots. The user tries to correct this by pressing the cursor-right key, and again overshoots. There is typically no way for the user to tell whether the buffer is empty or not, except by interacting very slowly with the system and observing that the cursor has moved after every keypress.

A similar problem, *icon wars*, occurs on window systems. The user clicks the mouse on a menu or icon, and nothing happens; for some reason the machine is busy or slow. So the user clicks again, tries something else – then, suddenly, all the buffered mouse clicks are interpreted and the screen becomes a blur of flashing windows and menus. This time, it is not so much that the response is too slow – it is fast enough when it happens – but that the response is variable. The delays due to swapping programs in and out of main memory typically cause these problems.

Furthermore, a style of interaction that is optimal on one machine may not be so on a slower machine. In particular, mouse-based interfaces cannot tolerate delays between actions and feedback of more than a fraction of a second, otherwise the immediacy required for successful interaction is lost. If these responses cannot be met then a more old-fashioned, command-based interface may be required.

Whereas it is immediately obvious that slow responses can cause problems for the user, it is not so obvious why one should not always aim for a system to be as fast as possible. However, there are exceptions to this – the user must be able to read and understand the output of the system. For example, one of the authors was once given a demonstration disk for a spreadsheet. Unfortunately, the machine the demo was written on was clearly slower than the author's machine, not much, at worst half the speed, but different enough. The demo passed in a blur over the screen with nothing remaining on the screen long enough to read. Many high-resolution monitors suffer from a similar problem when they display text. Whereas older character-based terminals scrolled new text from the bottom of the screen or redrew from the top, bitmap screens often 'flash' up the new page, giving no indication of direction of movement. A final example is the rate of cursor flashing: the rate is often at a fixed

Design Focus

The myth of the infinitely fast machine

The adverse effects of slow processing are made worse because the designers labor under the *myth of the infinitely fast machine* [93]. That is, they design and document their systems as if response will be immediate. Rather than blithely hoping that the eventual machine will be 'fast enough', the designer ought to plan explicitly for slow responses where these are possible. A good example, where buffering is clear and audible (if not visible) to the user, is telephones. Even if the user gets ahead of the telephone when entering a number, the tones can be heard as they are sent over the line. Now this is probably an accident of the design rather than deliberate policy, as there are so many other problems with telephones as interfaces. However, this type of serendipitous feedback should be emulated in other areas.

frequency, so varying the speed of the processor does not change the screen display. But a rate which is acceptable for a CRT screen is too fast for an LCD screen, which is more persistent, and the cursor may become invisible or a slight gray color.

In some ways the solution to these problems is easier: the designer can demand fixed delays (dependent on media and user preference) rather than just going as fast as the machine allows. To plan for the first problem, that of insufficient speed, the designer needs to understand the limitations of the computer system and take account of these at all stages in the design process.

2.9.2 Limitations on interactive performance

There are several factors that can limit the speed of an interactive system:

Computation bound This is rare for an interactive program, but possible, for example when using find/replace in a large document. The system should be designed so that long delays are not in the middle of interaction and so that the user gets some idea of how the job is progressing. For a very long process try to give an indication of duration *before* it starts; and during processing an indication of the stage that the process has reached is helpful. This can be achieved by having a counter or slowly filling bar on the screen that indicates the amount done, or by changing the cursor to indicate that processing is occurring. Many systems notice after they have been computing for some time and then say 'this may take some time: continue (Y/N)?'. Of course, by the time it says this the process may be nearly finished anyway!

Storage channel bound As we discussed in the previous section, the speed of memory access can interfere with interactive performance. We discussed one technique, laziness, for reducing this effect. In addition, if there is plenty of raw computation power and the system is held up solely by memory, it is possible to trade off memory against processing speed. For example, compressed data take less space to store, and is faster to read in and out, but must be compressed before storage and decompressed when retrieved. Thus faster memory access leads to increased processing time. If data is written more often than it is read, one can choose a technique that is expensive to compress but fairly simple to decompress. For many interactive systems the ability to browse quickly is very important, but users will accept delays when saving updated information.

Graphics bound For many modern interfaces, this is the most common bottleneck. It is easy to underestimate the time taken to perform what appear to be simple interface operations. Sometimes clever coding can reduce the time taken by common graphics operations, and there is tremendous variability in performance between programs running on the same hardware. Most computers include a special-purpose *graphics card* to handle many of the most common graphics operations. This is optimized for graphics operations and allows the main processor to do other work such as manipulating documents and other user data.

Network capacity Most computers are linked by networks. At the simplest this can mean using shared files on a remote machine. When accessing such files it can be the speed of the network rather than that of the memory which limits performance. This is discussed in greater detail below.

2.9.3 Networked computing

Computer systems in use today are much more powerful than they were a few years ago, which means that the standard computer on the desktop is quite capable of high-performance interaction without recourse to outside help. However, it is often the case that we use computers not in their standalone mode of operation, but linked together in networks. This brings added benefits in allowing communication between different parties, provided they are connected into the same network, as well as allowing the desktop computer to access resources remote from itself. Such networks are inherently much more powerful than the individual computers that make up the network: increased computing power and memory are only part of the story, since the effects of allowing people much more extensive, faster and easier access to information are highly significant to individuals, groups and institutions.

One of the biggest changes since the first edition of this book has been the explosive growth of the internet and global connectivity. As well as fixed networks it is now normal to use a high bandwidth modem or wireless local area network (LAN) to connect into the internet and world wide web from home or hotel room anywhere in the world. The effects of this on society at large can only be speculated upon at present, but there are already major effects on computer purchases and perhaps the whole face of personal computation. As more and more people buy computers principally to connect to the internet the idea of the *network computer* has arisen – a small computer with no disks whose sole purpose is to connect up to networks.

The internet

The internet has its roots back in 1969 as DARPANET when the US Government's Department of Defense commissioned research into networking. The initial four mainframe computers grew to 23 in 1971 and the system had been renamed ARPANET. Growth has accelerated ever since: in 1984 there were over a thousand machines connected, in 1989 the 100,000 mark had been reached, and the latest estimates are in the millions. All the computers on the system, now known as the internet, speak a set of common languages (protocols); the two most important of these are *Transmission Control Protocol* (*TCP*) which moves data from A to B, and the *Internet Protocol* (*IP*) which specifies which B is being referred to so that the data goes to the correct place. Together these protocols are known as *TCP/IP*. Thus, at its most basic level, the internet is simply millions of computers connected together and talking to each other. Other protocols then build on these low-level capabilities to provide services such as electronic mail, in which participants send messages to each other; news, where articles of interest are posted to a special interest group and can be read by anyone subscribing to that group; and of course the world wide web.

Such networked systems have an effect on interactivity, over and above any additional access to distant peripherals or information sources. Networks sometimes operate over large distances, and the transmission of information may take some appreciable time, which affects the response time of the system and hence the nature of the interactivity. There may be a noticeable delay in response, and if the user is not informed of what is going on, he may assume that his command has been ignored, or lost, and may then repeat it. This lack of feedback is an important factor in the poor performance and frustration users feel when using such systems, and can be alleviated by more sensible use of the capabilities of the desktop machine to inform users of what is happening over the network.

Another effect is that the interaction between human and machine becomes an open loop, rather than a closed one. Many people may be interacting with the machine at once, and their actions may affect the response to your own. Many users accessing a single central machine will slow its response; database updates carried out by one user may mean that the same query by another user at slightly different times may produce different results. The networked computer system, by the very nature of its dispersal, distribution and multi-user access, has been transformed from a fully predictable, deterministic system, under the total control of the user, into a non-deterministic one, with an individual user being unaware of many important things that are happening to the system as a whole. Such systems pose a particular problem since ideals of consistency, informative feedback and predictable response are violated (see Chapter 7 for more on these principles). However, the additional power and flexibility offered by networked systems means that they are likely to be with us for a long time, and these issues need to be carefully addressed in their design.

Worked exercise *How do you think new, fast, high-density memory devices and quick processors have influenced recent developments in HCI? Do they make systems any easier to use? Do they expand the range of applications of computer systems?*

Answer Arguably it is not so much the increase in computer power as the decrease in the cost of that power which has had the most profound effect. Because 'ordinary' users have powerful machines on their desktops it has become possible to view that power as available for the interface rather than hoarded for number-crunching applications.

Modern graphical interaction consumes vast amounts of processing power and would have been completely impossible only a few years ago. There is an extent to which systems have to run faster to stay still, in that as screen size, resolution and color range increase, so does the necessary processing power to maintain the 'same' interaction. However, this extra processing is not really producing the same effect; screen quality is still a major block on effective interaction.

The increase in RAM means that larger programs can be written, effectively allowing the programmer 'elbow room'. This is used in two ways: to allow extra functionality and to support easier interaction. Whether the former really improves usability is debatable – unused functionality is a good marketing point, but is of no benefit to the user. The ease of use of a system is often determined by a host of small features, such as the appropriate choice of default options. These features make the interface seem 'simple', but make the program very complex . . . and large. Certainly the availability of elbow room, both in terms of memory and processing power, has made such features possible.

The increase in both short-term (RAM) and long-term (disks and optical storage) memory has also removed many of the arbitrary limits in systems: it is possible to edit documents of virtually unlimited size and to treat the computer (suitably backed up) as one's primary information repository.

Some whole new application areas have become possible because of advances in memory and processing. Most applications of multimedia including voice recognition and online storage and capture of video and audio, require enormous amounts of processing and/or memory. In particular, large magnetic and optical storage devices have been the key to electronic document storage whereby all paper documents are scanned and stored within a computer system. In some contexts such systems have completely replaced paper-based filing cabinets.

2.10 SUMMARY

In Sections 2.2 and 2.3, we described a range of input devices. These performed two main functions: text entry and pointing. The principal text entry device is the QWERTY keyboard, but we also discussed alternative keyboards, chord keyboards, the telephone keypad and speech input. Pointing devices included the mouse, touchpad, trackball and joystick, as well as a large array of less common alternatives including eyegaze systems.

Section 2.4 dealt mainly with the screen as a direct output device. We discussed several different technologies, in particular CRT and LCD screens and the common properties of all bitmap display devices. We considered some more recent display methods including large displays, situated displays and digital paper.

Section 2.5 looked at the devices used for manipulating and seeing virtual reality and 3D spaces. This included the dataglove, body tracking, head-mounted displays and cave environments.

In Section 2.6 we moved outside the computer entirely and looked at physical devices such as the special displays, knobs and switches of electronic appliances. We also briefly considered sound, touch and smell as outputs from computer systems and environmental and bio-sensing as inputs. These are topics that will be revisited later in the book.

Section 2.7 discussed various forms of printer and scanner. Typical office printers include ink-jet, bubble-jet and laser printers. In addition, dot-matrix and thermal printers are used in specialized equipment. We also discussed font styles and page description languages. Scanners are used to convert printed images and documents into electronic form. They are particularly valuable in desktop publishing and for electronic document storage systems.

In Section 2.8, we considered the typical capacities of computer memory, both of main RAM, likened to human short-term memory, and long-term memory stored on magnetic and optical disks. The storage capacities were compared with document sizes and video images. We saw that a typical hard disk could only hold about two minutes of moving video, but that compression techniques can increase the capacity dramatically. We also discussed storage standards – or rather the lack of them – including the ASCII character set and markup languages. The user ought to be able to access information in ways that are natural and tolerant of small slips. Techniques which can help this included multiple indices, free text databases, DWIM (do what I mean) and Soundex.

Section 2.9 showed how processing speed, whether too slow or too fast, can affect the user interface. In particular, we discussed the effects of buffering: cursor tracking and icon wars. Processing speed is limited by various factors: computation, memory access, graphics and network delays.

The lesson from this chapter is that the interface designer needs to be aware of the properties of the devices with which a system is built. This includes not only input and output devices, but all the factors that influence the behavior of the interface, since all of these influence the nature and style of the interaction.

EXERCISES

2.1 Individually or in a group find as many different examples as you can of physical controls and displays.

(a) List them.
(b) Try to group them, or classify them.
(c) Discuss whether you believe the control or display is suitable for its purpose (Section 3.9.3 may also help).

Exercises 2.2 and 2.3 involve you examining a range of input and output devices in order to understand how they influence interaction.

2.2 A typical computer system comprises a QWERTY keyboard, a mouse and a color screen. There is usually some form of loudspeaker as well. You should know how the keyboard, mouse and screen work – if not, read up on it.

What sort of input does the keyboard support? What sort of input does the mouse support? Are these adequate for all possible applications? If not, to which areas are they most suited? Do these areas map well onto the typical requirements for users of computer systems?

If you were designing a keyboard for a modern computer, and you wanted to produce a faster, easier-to-use layout, what information would you need to know and how would that influence the design?

2.3 Pick a couple of computer input devices that you are aware of (joystick, light pen, touchscreen, trackball, eyegaze, dataglove, etc.) and note down how each has different attributes that support certain forms of interaction. You ought to know a little about all of these devices – if you don't, research them.

2.4 What is the myth of the infinitely fast machine?

2.5 Pick one of the following scenarios, and choose a suitable combination of input and output devices to best support the intended interaction. It may help to identify typical users or classes of user, and identify how the devices chosen support these people in their tasks. Explain the major problems that the input and output devices solve.

(a) *Environmental database*
A computer database is under development that will hold environmental information. This ranges from meteorological measurements through fish catches to descriptions of pollution, and will include topographical details and sketches and photographs. The data has to be accessed only by experts, but they want to be able to describe and retrieve any piece of data within a few seconds.

(b) *Word processor for blind people*
A word processor for blind users is needed, which can also be operated by sighted people. It has to support the standard set of word-processing tasks.

2.6 Describe Fitts' law (see Chapter 1). How does Fitts' law change for different physical selection devices, such as a three-button mouse, a touchpad, or a pen/stylus? (You'll need to do some research for this.)

RECOMMENDED READING

W. Buxton, There's more to interaction than meets the eye: some issues in manual input. In R. Baecker and W. Buxton, editors, *Readings in Human–Computer Interaction: A Multidisciplinary Approach*, Morgan Kaufmann, 1987.

D. J. Mayhew, *Principles and Guidelines in Software User Interface Design*, Chapter 12, Prentice Hall, 1992.
A look at input and output devices, complete with guidelines for using different devices.

A. Dix, Network-based interaction. In J. Jacko and A. Sears, editors, *Human–Computer Interaction Handbook*, Chapter 16, pp. 331–57, Lawrence Erlbaum, 2003.
Includes different kinds of network application and the effects of networks on interaction including rich media. Also look at all of Part II of Jacko and Sears, which includes chapters on input technology and on haptic interfaces.

3

THE INTERACTION

OVERVIEW

- Interaction models help us to understand what is going on in the interaction between user and system. They address the translations between what the user wants and what the system does.
- Ergonomics looks at the physical characteristics of the interaction and how these influence its effectiveness.
- The dialog between user and system is influenced by the style of the interface.
- The interaction takes place within a social and organizational context that affects both user and system.

3.1 INTRODUCTION

In the previous two chapters we have looked at the human and the computer respectively. However, in the context of this book, we are not concerned with them in isolation. We are interested in how the human user uses the computer as a tool to perform, simplify or support a task. In order to do this the user must communicate his requirements to the computer.

There are a number of ways in which the user can communicate with the system. At one extreme is batch input, in which the user provides all the information to the computer at once and leaves the machine to perform the task. This approach does involve an interaction between the user and computer but does not support many tasks well. At the other extreme are highly interactive input devices and paradigms, such as *direct manipulation* (see Chapter 4) and the applications of *virtual reality* (Chapter 20). Here the user is constantly providing instruction and receiving feedback. These are the types of interactive system we are considering.

In this chapter, we consider the communication between user and system: the *interaction*. We will look at some models of interaction that enable us to identify and evaluate components of the interaction, and at the physical, social and organizational issues that provide the context for it. We will also survey some of the different styles of interaction that are used and consider how well they support the user.

3.2 MODELS OF INTERACTION

In previous chapters we have seen the usefulness of models to help us to understand complex behavior and complex systems. Interaction involves at least two participants: the user and the system. Both are complex, as we have seen, and are very different from each other in the way that they communicate and view the domain and the task. The interface must therefore effectively translate between them to allow the interaction to be successful. This translation can fail at a number of points and for a number of reasons. The use of models of interaction can help us to understand exactly what is going on in the interaction and identify the likely root of difficulties. They also provide us with a framework to compare different interaction styles and to consider interaction problems.

We begin by considering the most influential model of interaction, Norman's *execution–evaluation cycle*; then we look at another model which extends the ideas of Norman's cycle. Both of these models describe the interaction in terms of the goals and actions of the user. We will therefore briefly discuss the terminology used and the assumptions inherent in the models, before describing the models themselves.

3.2.1 The terms of interaction

Traditionally, the purpose of an interactive system is to aid a user in accomplishing *goals* from some application *domain*. (Later in this book we will look at alternative interactions but this model holds for many work-oriented applications.) A domain defines an area of expertise and knowledge in some real-world activity. Some examples of domains are graphic design, authoring and process control in a factory. A domain consists of concepts that highlight its important aspects. In a graphic design domain, some of the important concepts are geometric shapes, a drawing surface and a drawing utensil. *Tasks* are operations to manipulate the concepts of a domain. A *goal* is the desired output from a performed task. For example, one task within the graphic design domain is the construction of a specific geometric shape with particular attributes on the drawing surface. A related goal would be to produce a solid red triangle centered on the canvas. An *intention* is a specific action required to meet the goal.

Task analysis involves the identification of the problem space (which we discussed in Chapter 1) for the user of an interactive system in terms of the domain, goals, intentions and tasks. We can use our knowledge of tasks and goals to assess the interactive system that is designed to support them. We discuss task analysis in detail in Chapter 15. The concepts used in the design of the system and the description of the user are separate, and so we can refer to them as distinct components, called the *System* and the *User*, respectively. The *System* and *User* are each described by means of a language that can express concepts relevant in the domain of the application. The *System*'s language we will refer to as the *core language* and the *User*'s language we will refer to as the *task language*. The core language describes computational attributes of the domain relevant to the *System* state, whereas the task language describes psychological attributes of the domain relevant to the *User* state.

The system is assumed to be some computerized application, in the context of this book, but the models apply equally to non-computer applications. It is also a common assumption that by distinguishing between user and system we are restricted to single-user applications. This is not the case. However, the emphasis is on the view of the interaction from a single user's perspective. From this point of view, other users, such as those in a multi-party conferencing system, form part of the system.

3.2.2 The execution–evaluation cycle

Norman's model of interaction is perhaps the most influential in Human–Computer Interaction, possibly because of its closeness to our intuitive understanding of the interaction between human user and computer [265]. The user formulates a plan of action, which is then executed at the computer interface. When the plan, or part of the plan, has been executed, the user observes the computer interface to evaluate the result of the executed plan, and to determine further actions.

The interactive cycle can be divided into two major phases: execution and evaluation. These can then be subdivided into further stages, seven in all. The stages in Norman's model of interaction are as follows:

1. Establishing the goal.
2. Forming the intention.
3. Specifying the action sequence.
4. Executing the action.
5. Perceiving the system state.
6. Interpreting the system state.
7. Evaluating the system state with respect to the goals and intentions.

Each stage is, of course, an activity of the user. First the user forms a goal. This is the user's notion of what needs to be done and is framed in terms of the domain, in the task language. It is liable to be imprecise and therefore needs to be translated into the more specific intention, and the actual actions that will reach the goal, before it can be executed by the user. The user perceives the new state of the system, after execution of the action sequence, and interprets it in terms of his expectations. If the system state reflects the user's goal then the computer has done what he wanted and the interaction has been successful; otherwise the user must formulate a new goal and repeat the cycle.

Norman uses a simple example of switching on a light to illustrate this cycle. Imagine you are sitting reading as evening falls. You decide you need more light; that is you establish the goal to get more light. From there you form an intention to switch on the desk lamp, and you specify the actions required, to reach over and press the lamp switch. If someone else is closer the intention may be different – you may ask them to switch on the light for you. Your goal is the same but the intention and actions are different. When you have executed the action you perceive the result, either the light is on or it isn't and you interpret this, based on your knowledge of the world. For example, if the light does not come on you may interpret this as indicating the bulb has blown or the lamp is not plugged into the mains, and you will formulate new goals to deal with this. If the light does come on, you will evaluate the new state according to the original goals – is there now enough light? If so, the cycle is complete. If not, you may formulate a new intention to switch on the main ceiling light as well.

Norman uses this model of interaction to demonstrate why some interfaces cause problems to their users. He describes these in terms of the *gulfs of execution* and the *gulfs of evaluation*. As we noted earlier, the user and the system do not use the same terms to describe the domain and goals – remember that we called the language of the system the *core language* and the language of the user the *task language*. The gulf of execution is the difference between the user's formulation of the actions to reach the goal and the actions allowed by the system. If the actions allowed by the system correspond to those intended by the user, the interaction will be effective. The interface should therefore aim to reduce this gulf.

The gulf of evaluation is the distance between the physical presentation of the system state and the expectation of the user. If the user can readily evaluate the presentation in terms of his goal, the gulf of evaluation is small. The more effort that is required on the part of the user to interpret the presentation, the less effective the interaction.

Human error – slips and mistakes

Human errors are often classified into *slips* and *mistakes*. We can distinguish these using Norman's gulf of execution.

If you understand a system well you may know exactly what to do to satisfy your goals – you have formulated the correct action. However, perhaps you mistype or you accidentally press the mouse button at the wrong time. These are called *slips*; you have formulated the right action, but fail to execute that action correctly.

However, if you don't know the system well you may not even formulate the right goal. For example, you may think that the magnifying glass icon is the 'find' function, but in fact it is to magnify the text. This is called a *mistake*.

If we discover that an interface is leading to errors it is important to understand whether they are slips or mistakes. Slips may be corrected by, for instance, better screen design, perhaps putting more space between buttons. However, mistakes need users to have a better understanding of the systems, so will require far more radical redesign or improved training, perhaps a totally different metaphor for use.

Norman's model is a useful means of understanding the interaction, in a way that is clear and intuitive. It allows other, more detailed, empirical and analytic work to be placed within a common framework. However, it only considers the system as far as the interface. It concentrates wholly on the user's view of the interaction. It does not attempt to deal with the system's communication through the interface. An extension of Norman's model, proposed by Abowd and Beale, addresses this problem [3]. This is described in the next section.

3.2.3 The interaction framework

The interaction framework attempts a more realistic description of interaction by including the system explicitly, and breaks it into four main components, as shown in Figure 3.1. The nodes represent the four major components in an interactive system – the *System*, the *User*, the *Input* and the *Output*. Each component has its own language. In addition to the *User*'s task language and the *System*'s core language, which we have already introduced, there are languages for both the *Input* and *Output* components. *Input* and *Output* together form the *Interface*.

As the interface sits between the *User* and the *System*, there are four steps in the interactive cycle, each corresponding to a translation from one component to another, as shown by the labeled arcs in Figure 3.2. The *User* begins the interactive cycle with the formulation of a goal and a task to achieve that goal. The only way the user can manipulate the machine is through the *Input*, and so the task must be articulated within the input language. The input language is translated into the core

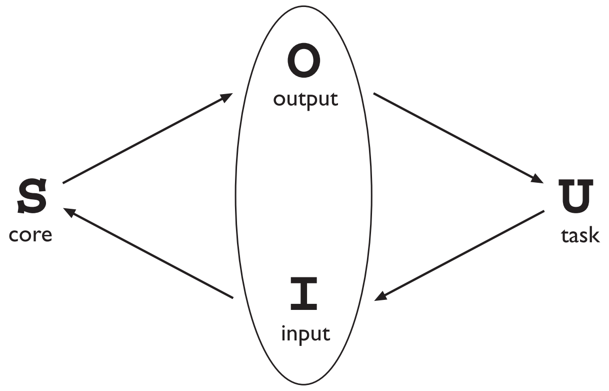


Figure 3.1 The general interaction framework

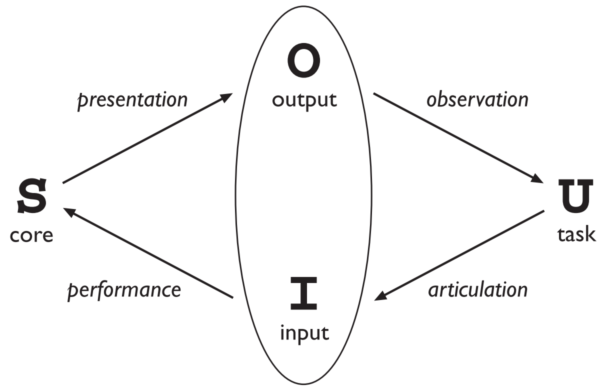


Figure 3.2 Translations between components

language as operations to be performed by the *System*. The *System* then transforms itself as described by the operations; the execution phase of the cycle is complete and the evaluation phase now begins. The *System* is in a new state, which must now be communicated to the *User*. The current values of system attributes are rendered as concepts or features of the *Output*. It is then up to the *User* to observe the *Output* and assess the results of the interaction relative to the original goal, ending the evaluation phase and, hence, the interactive cycle. There are four main translations involved in the interaction: articulation, performance, presentation and observation.

The *User*'s formulation of the desired task to achieve some goal needs to be *articulated* in the input language. The tasks are responses of the *User* and they need to be translated to stimuli for the *Input*. As pointed out above, this articulation is judged in terms of the coverage from tasks to input and the relative ease with which the translation can be accomplished. The task is phrased in terms of certain psychological attributes that highlight the important features of the domain for the *User*. If these psychological attributes map clearly onto the input language, then articulation of the task will be made much simpler. An example of a poor mapping, as pointed out by Norman, is a large room with overhead lighting controlled by a bank of switches. It is often desirable to control the lighting so that only one section of the room is lit. We are then faced with the puzzle of determining which switch controls which lights. The result is usually repeated trials and frustration. This arises from the difficulty of articulating a goal (for example, 'Turn on the lights in the front of the room') in an input language that consists of a linear row of switches, which may or may not be oriented to reflect the room layout.

Conversely, an example of a good mapping is in virtual reality systems, where input devices such as datagloves are specifically geared towards easing articulation by making the user's psychological notion of gesturing an act that can be directly realized at the interface. Direct manipulation interfaces, such as those found on common desktop operating systems like the Macintosh and Windows, make the articulation of some file handling commands easier. On the other hand, some tasks, such as repetitive file renaming or launching a program whose icon is not visible, are not at all easy to articulate with such an interface.

At the next stage, the responses of the *Input* are translated to stimuli for the *System*. Of interest in assessing this translation is whether the translated input language can reach as many states of the *System* as is possible using the *System* stimuli directly. For example, the remote control units for some compact disc players do not allow the user to turn the power off on the player unit; hence the off state of the player cannot be reached using the remote control's input language. On the panel of the compact disc player, however, there is usually a button that controls the power. The ease with which this translation from *Input* to *System* takes place is of less importance because the effort is not expended by the user. However, there can be a real effort expended by the designer and programmer. In this case, the ease of the translation is viewed in terms of the cost of implementation.

Once a state transition has occurred within the *System*, the execution phase of the interaction is complete and the evaluation phase begins. The new state of the *System* must be communicated to the *User*, and this begins by translating the *System* responses to the transition into stimuli for the *Output* component. This presentation translation must preserve the relevant system attributes from the domain in the limited expressiveness of the output devices. The ability to capture the domain concepts of the *System* within the *Output* is a question of expressiveness for this translation.

For example, while writing a paper with some word-processing package, it is necessary at times to see both the immediate surrounding text where one is currently composing, say, the current paragraph, and a wider context within the whole paper that cannot be easily displayed on one screen (for example, the current chapter).

Ultimately, the user must interpret the output to evaluate what has happened. The response from the *Output* is translated to stimuli for the *User* which trigger assessment. The observation translation will address the ease and coverage of this final translation. For example, it is difficult to tell the time accurately on an unmarked analog clock, especially if it is not oriented properly. It is difficult in a command line interface to determine the result of copying and moving files in a hierarchical file system. Developing a website using a markup language like HTML would be virtually impossible without being able to preview the output through a browser.

Assessing overall interaction

The interaction framework is presented as a means to judge the overall usability of an entire interactive system. In reality, all of the analysis that is suggested by the framework is dependent on the current task (or set of tasks) in which the *User* is engaged. This is not surprising since it is only in attempting to perform a particular task within some domain that we are able to determine if the tools we use are adequate. For example, different text editors are better at different things. For a particular editing task, one can choose the text editor best suited for interaction relative to the task. The best editor, if we are forced to choose only one, is the one that best suits the tasks most frequently performed. Therefore, it is not too disappointing that we cannot extend the interaction analysis beyond the scope of a particular task.

Design Focus

Video recorder

A simple example of programming a VCR from a remote control shows that all four translations in the interaction cycle can affect the overall interaction. Ineffective interaction is indicated by the user not being sure the VCR is set to record properly. This could be because the user has pressed the keys on the remote control unit in the wrong order; this can be classified as an articulatory problem. Or maybe the VCR is able to record on any channel but the remote control lacks the ability to select channels, indicating a coverage problem for the performance translation. It may be the case that the VCR display panel does not indicate that the program has been set, a presentation problem. Or maybe the user does not interpret the feedback properly, an observational error. Any one or more of these deficiencies would give rise to ineffective interaction.

3.3 FRAMEWORKS AND HCI

As well as providing a means of discussing the details of a particular interaction, frameworks provide a basis for discussing other issues that relate to the interaction. The ACM SIGCHI Curriculum Development Group presents a framework similar to that presented here, and uses it to place different areas that relate to HCI [9].

In Figure 3.3 these aspects are shown as they relate to the interaction framework. In particular, the field of *ergonomics* addresses issues on the user side of the interface, covering both input and output, as well as the user's immediate context. Dialog design and interface styles can be placed particularly along the input branch of the framework, addressing both articulation and performance. However, dialog is most usually associated with the computer and so is biased to that side of the framework.

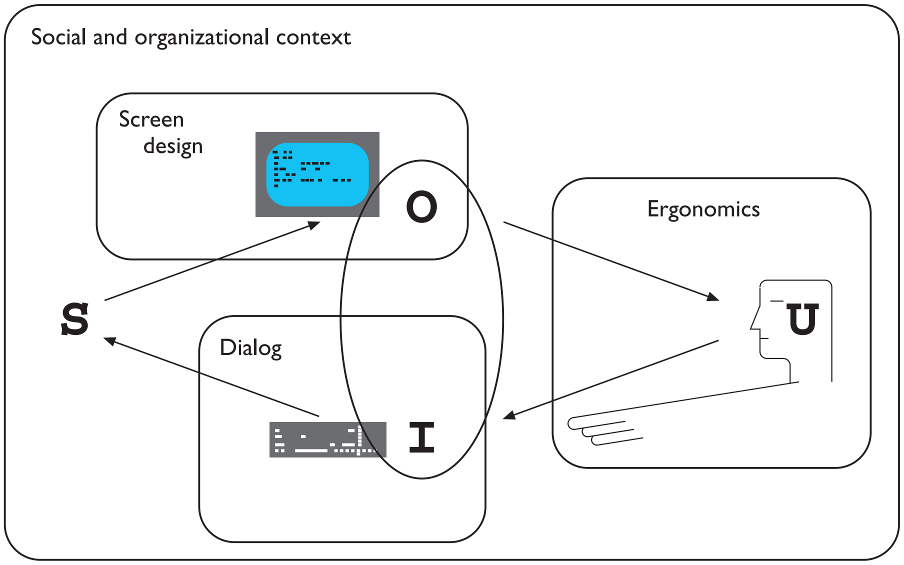


Figure 3.3 A framework for human–computer interaction. Adapted from ACM SIGCHI Curriculum Development Group [9]

Presentation and screen design relates to the output branch of the framework. The entire framework can be placed within a social and organizational context that also affects the interaction. Each of these areas has important implications for the design of interactive systems and the performance of the user. We will discuss these in brief in the following sections, with the exception of screen design which we will save until Chapter 5.

3.4 ERGONOMICS

Ergonomics (or human factors) is traditionally the study of the physical characteristics of the interaction: how the controls are designed, the physical environment in which the interaction takes place, and the layout and physical qualities of the screen. A primary focus is on user performance and how the interface enhances or detracts from this. In seeking to evaluate these aspects of the interaction, ergonomics will certainly also touch upon human psychology and system constraints. It is a large and established field, which is closely related to but distinct from HCI, and full coverage would demand a book in its own right. Here we consider a few of the issues addressed by ergonomics as an introduction to the field. We will briefly look at the arrangement of controls and displays, the physical environment, health issues and the use of color. These are by no means exhaustive and are intended only to give an indication of the types of issues and problems addressed by ergonomics. For more information on ergonomic issues the reader is referred to the recommended reading list at the end of the chapter.

3.4.1 Arrangement of controls and displays

In Chapter 1 we considered perceptual and cognitive issues that affect the way we present information on a screen and provide control mechanisms to the user. In addition to these cognitive aspects of design, physical aspects are also important. Sets of controls and parts of the display should be grouped logically to allow rapid access by the user (more on this in Chapter 5). This may not seem so important when we are considering a single user of a spreadsheet on a PC, but it becomes vital when we turn to safety-critical applications such as plant control, aviation and air traffic control. In each of these contexts, users are under pressure and are faced with a huge range of displays and controls. Here it is crucial that the physical layout of these be appropriate. Indeed, returning to the less critical PC application, inappropriate placement of controls and displays can lead to inefficiency and frustration. For example, on one particular electronic newsreader, used by one of the authors, the command key to read articles from a newsgroup (y) is directly beside the command key to unsubscribe from a newsgroup (u) on the keyboard. This poor design frequently leads to inadvertent removal of newsgroups. Although this is recoverable it wastes time and is annoying to the user. We saw similar examples in the Introduction to this book including the MacOS X dock. We can therefore see that appropriate layout is important in all applications.

We have already touched on the importance of grouping controls together logically (and keeping opposing controls separate). The exact organization that this will suggest will depend on the domain and the application, but possible organizations include the following:

functional controls and displays are organized so that those that are functionally related are placed together;

sequential controls and displays are organized to reflect the order of their use in a typical interaction (this may be especially appropriate in domains where a particular task sequence is enforced, such as aviation);

frequency controls and displays are organized according to how frequently they are used, with the most commonly used controls being the most easily accessible.

In addition to the organization of the controls and displays in relation to each other, the entire system interface must be arranged appropriately in relation to the user's position. So, for example, the user should be able to reach all controls necessary and view all displays without excessive body movement. Critical displays should be at eye level. Lighting should be arranged to avoid glare and reflection distorting displays. Controls should be spaced to provide adequate room for the user to manoeuvre.

DESIGN FOCUS

Industrial interfaces

The interfaces to office systems have changed dramatically since the 1980s. However, some care is needed in transferring the idioms of office-based systems into the industrial domain. Office information is primarily textual and slow varying, whereas industrial interfaces may require the rapid assimilation of multiple numeric displays, each of which is varying in response to the environment. Furthermore, the environmental conditions may rule out certain interaction styles (for example, the oil-soaked mouse). Consequently, industrial interfaces raise some additional design issues rarely encountered in the office.

Glass interfaces vs. dials and knobs
The traditional machine interface consists of dials and knobs directly wired or piped to the equipment. Increasingly, some or all of the controls are replaced with a glass interface, a computer screen through which the equipment is monitored and controlled. Many of the issues are similar for the two kinds of interface, but glass interfaces do have some special advantages and problems. For a complex system, a glass interface can be both cheaper and more flexible, and it is easy to show the same information in multiple forms (Figure 3.4). For example, a data value might be given both in a precise numeric field and also in a quick to assimilate graphical form. In addition, the same information can be shown on several screens. However, the information is not located in physical space and so vital clues to context are missing – it is easy to get lost navigating complex menu systems. Also, limited display resolution often means that an electronic representation of a dial is harder to read than its physical counterpart; in some circumstances both may be necessary, as is the case on the flight deck of a modern aircraft.

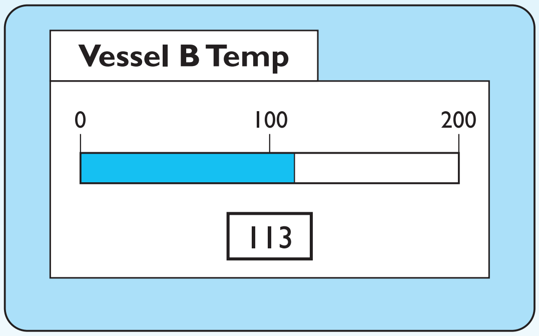


Figure 3.4 Multiple representations of the same information

Indirect manipulation
The phrase 'direct manipulation' dominates office system design (Figure 3.5). There are arguments about its meaning and appropriateness even there, but it is certainly dependent on the user being in primary control of the changes in the interface. The autonomous nature of industrial processes makes this an inappropriate model. In a direct manipulation system, the user interacts with an artificial world inside the computer (for example, the electronic desktop).

In contrast, an industrial interface is merely an intermediary between the operator and the real world. One implication of this indirectness is that the interface must provide feedback at two levels

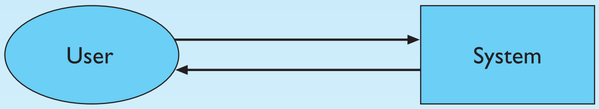


Figure 3.5 Office system – direct manipulation

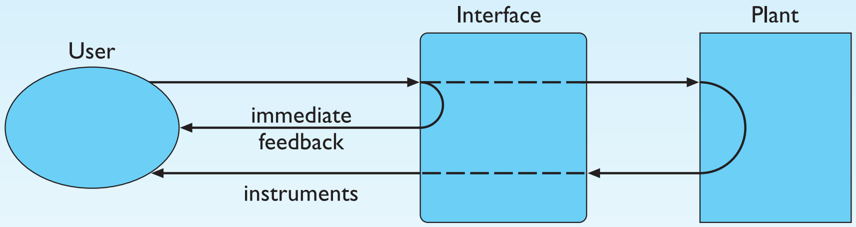


Figure 3.6 Indirect manipulation – two kinds of feedback

(Figure 3.6). At one level, the user must receive immediate feedback, generated by the interface, that keystrokes and other actions have been received. In addition, the user's actions will have some effect on the equipment controlled by the interface and adequate monitoring must be provided for this.

The indirectness also causes problems with simple monitoring tasks. Delays due to periodic sampling, slow communication and digital processing often mean that the data displayed are somewhat out of date. If the operator is not aware of these delays, diagnoses of system state may be wrong. These problems are compounded if the interface produces summary information displays. If the data comprising such a display are of different timeliness the result may be misleading.

3.4.2 The physical environment of the interaction

As well as addressing physical issues in the layout and arrangement of the machine interface, ergonomics is concerned with the design of the work environment itself. Where will the system be used? By whom will it be used? Will users be sitting, standing or moving about? Again, this will depend largely on the domain and will be more critical in specific control and operational settings than in general computer use. However, the physical environment in which the system is used may influence how well it is accepted and even the health and safety of its users. It should therefore be considered in all design.

The first consideration here is the size of the users. Obviously this is going to vary considerably. However, in any system the smallest user should be able to reach all the controls (this may include a user in a wheelchair), and the largest user should not be cramped in the environment.

In particular, all users should be comfortably able to see critical displays. For long periods of use, the user should be seated for comfort and stability. Seating should provide back support. If required to stand, the user should have room to move around in order to reach all the controls.

3.4.3 Health issues

Perhaps we do not immediately think of computer use as a hazardous activity but we should bear in mind possible consequences of our designs on the health and safety of users. Leaving aside the obvious safety risks of poorly designed safety-critical systems (aircraft crashing, nuclear plant leaks and worse), there are a number of factors that may affect the use of more general computers. Again these are factors in the physical environment that directly affect the quality of the interaction and the user's performance:

Physical position As we noted in the previous section, users should be able to reach all controls comfortably and see all displays. Users should not be expected to stand for long periods and, if sitting, should be provided with back support. If a particular position for a part of the body is to be adopted for long periods (for example, in typing) support should be provided to allow rest.

Temperature Although most users can adapt to slight changes in temperature without adverse effect, extremes of hot or cold will affect performance and, in excessive cases, health. Experimental studies show that performance deteriorates at high or low temperatures, with users being unable to concentrate efficiently.

Lighting The lighting level will again depend on the work environment. However, adequate lighting should be provided to allow users to see the computer screen without discomfort or eyestrain. The light source should also be positioned to avoid glare affecting the display.

Noise Excessive noise can be harmful to health, causing the user pain, and in acute cases, loss of hearing. Noise levels should be maintained at a comfortable level in the work environment. This does not necessarily mean no noise at all. Noise can be a stimulus to users and can provide needed confirmation of system activity.

Time The time users spend using the system should also be controlled. As we saw in the previous chapter, it has been suggested that excessive use of CRT displays can be harmful to users, particularly pregnant women.

3.4.4 The use of color

In this section we have concentrated on the ergonomics of physical characteristics of systems, including the physical environment in which they are used. However, ergonomics has a close relationship to human psychology in that it is also concerned with the perceptual limitations of humans. For example, the use of color in displays is an ergonomics issue. As we saw in Chapter 1, the visual system has some limitations with regard to color, including the number of colors that are distinguishable and the relatively low blue acuity. We also saw that a relatively high proportion of the population suffers from a deficiency in color vision. Each of these psychological phenomena leads to ergonomic guidelines; some examples are discussed below.

Colors used in the display should be as distinct as possible and the distinction should not be affected by changes in contrast. Blue should not be used to display critical information. If color is used as an indicator it should not be the only cue: additional coding information should be included.

The colors used should also correspond to common conventions and user expectations. Red, green and yellow are colors frequently associated with stop, go and standby respectively. Therefore, red may be used to indicate emergency and alarms; green, normal activity; and yellow, standby and auxiliary function. These conventions should not be violated without very good cause.

However, we should remember that color conventions are culturally determined. For example, red is associated with danger and warnings in most western cultures, but in China it symbolizes happiness and good fortune. The color of mourning is black in some cultures and white in others. Awareness of the cultural associations of color is particularly important in designing systems and websites for a global market. We will return to these issues in more detail in Chapter 10.

3.4.5 Ergonomics and HCI

Ergonomics is a huge area, which is distinct from HCI but sits alongside it. Its contribution to HCI is in determining constraints on the way we design systems and suggesting detailed and specific guidelines and standards. Ergonomic factors are in general well established and understood and are therefore used as the basis for standardizing hardware designs. This issue is discussed further in Chapter 7.

3.5 INTERACTION STYLES

Interaction can be seen as a dialog between the computer and the user. The choice of interface style can have a profound effect on the nature of this dialog. Dialog design is discussed in detail in Chapter 16. Here we introduce the most common interface styles and note the different effects these have on the interaction. There are a number of common interface styles including

- command line interface
- menus
- natural language
- question/answer and query dialog
- form-fills and spreadsheets
- WIMP
- point and click
- three-dimensional interfaces.

As the WIMP interface is the most common and complex, we will discuss each of its elements in greater detail in Section 3.6.

```
sable.soc.staffs.ac.uk> javac HelloWorldApp
javac: invalid argument: HelloWorldApp
use: javac [-g][-O][-classpath path][-d dir] file.java…
sable.soc.staffs.ac.uk> javac HelloWorldApp.java
sable.soc.staffs.ac.uk> java HelloWorldApp
Hello world!!
sable.soc.staffs.ac.uk>
```

Figure 3.7 Command line interface

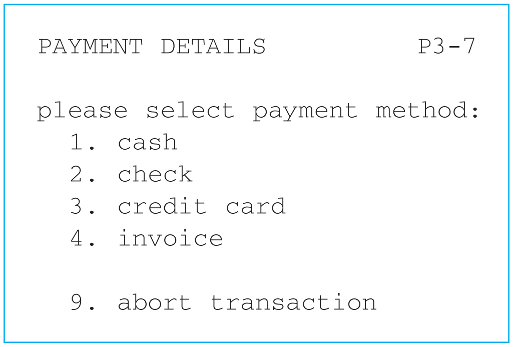
3.5.1 Command line interface

The command line interface (Figure 3.7) was the first interactive dialog style to be commonly used and, in spite of the availability of menu-driven interfaces, it is still widely used. It provides a means of expressing instructions to the computer directly, using function keys, single characters, abbreviations or whole-word commands. In some systems the command line is the only way of communicating with the system, especially for remote access using *telnet*. More commonly today it is supplementary to menu-based interfaces, providing accelerated access to the system's functionality for experienced users.

Command line interfaces are powerful in that they offer direct access to system functionality (as opposed to the hierarchical nature of menus), and can be combined to apply a number of tools to the same data. They are also flexible: the command often has a number of options or parameters that will vary its behavior in some way, and it can be applied to many objects at once, making it useful for repetitive tasks. However, this flexibility and power brings with it difficulty in use and learning. Commands must be remembered, as no cue is provided in the command line to indicate which command is needed. They are therefore better for expert users than for novices. This problem can be alleviated a little by using consistent and meaningful commands and abbreviations. The commands used should be terms within the vocabulary of the user rather than the technician. Unfortunately, commands are often obscure and vary across systems, causing confusion to the user and increasing the overhead of learning.

3.5.2 Menus

In a menu-driven interface, the set of options available to the user is displayed on the screen, and selected using the mouse, or numeric or alphabetic keys. Since the options are visible they are less demanding of the user, relying on recognition rather than recall. However, menu options still need to be meaningful and logically grouped to aid recognition. Often menus are hierarchically ordered and the option required is not available at the top layer of the hierarchy. The grouping



```
PAYMENT DETAILS               P3-7

please select payment method:
   1. cash
   2. check
   3. credit card
   4. invoice

   9. abort transaction
```

Figure 3.8 Menu-driven interface

and naming of menu options then provides the only cue for the user to find the required option. Such systems either can be purely text based, with the menu options being presented as numbered choices (see Figure 3.8), or may have a graphical component in which the menu appears within a rectangular box and choices are made, perhaps by typing the initial letter of the desired selection, or by entering the associated number, or by moving around the menu with the arrow keys. This is a restricted form of a full WIMP system, described in more detail shortly.

3.5.3 Natural language

Perhaps the most attractive means of communicating with computers, at least at first glance, is by natural language. Users, unable to remember a command or lost in a hierarchy of menus, may long for the computer that is able to understand instructions expressed in everyday words! Natural language understanding, both of speech and written input, is the subject of much interest and research. Unfortunately, however, the ambiguity of natural language makes it very difficult for a machine to understand. Language is ambiguous at a number of levels. First, the syntax, or structure, of a phrase may not be clear. If we are given the sentence

```
The boy hit the dog with the stick
```

we cannot be sure whether the boy is using the stick to hit the dog or whether the dog is holding the stick when it is hit.

Even if a sentence's structure is clear, we may find ambiguity in the meaning of the words used. For example, the word 'pitch' may refer to a sports field, a throw, a waterproofing substance or even, colloquially, a territory. We often rely on the context and our general knowledge to sort out these ambiguities. This information is difficult to provide to the machine. To complicate matters more, the use of pronouns and relative terms adds further ambiguity.

Given these problems, it seems unlikely that a general natural language interface will be available for some time. However, systems can be built to understand restricted subsets of a language. For a known and constrained domain, the system can be provided with sufficient information to disambiguate terms. It is important in interfaces which use natural language in this restricted form that the user is aware of the limitations of the system and does not expect too much understanding.

The use of natural language in restricted domains is relatively successful, but it is debatable whether this can really be called natural language. The user still has to learn which phrases the computer understands and may become frustrated if too much is expected. However, it is also not clear how useful a general natural language interface would be. Language is by nature vague and imprecise: this gives it its flexibility and allows creativity in expression. Computers, on the other hand, require precise instructions. Given a free rein, would we be able to describe our requirements precisely enough to guarantee a particular response? And, if we could, would the language we used turn out to be a restricted subset of natural language anyway?

3.5.4 Question/answer and query dialog

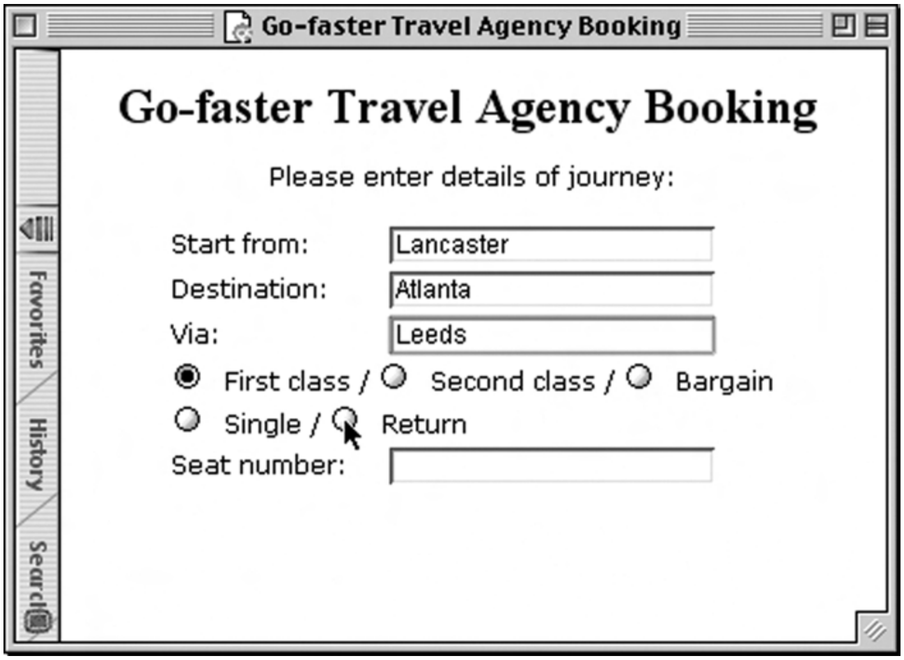
Question and answer dialog is a simple mechanism for providing input to an application in a specific domain. The user is asked a series of questions (mainly with yes/no responses, multiple choice, or codes) and so is led through the interaction step by step. An example of this would be web questionnaires.

These interfaces are easy to learn and use, but are limited in functionality and power. As such, they are appropriate for restricted domains (particularly information systems) and for novice or casual users.

Query languages, on the other hand, are used to construct queries to retrieve information from a database. They use natural-language-style phrases, but in fact require specific syntax, as well as knowledge of the database structure. Queries usually require the user to specify an attribute or attributes for which to search the database, as well as the attributes of interest to be displayed. This is straightforward where there is a single attribute, but becomes complex when multiple attributes are involved, particularly if the user is interested in attribute A or attribute B, or attribute A and not attribute B, or where values of attributes are to be compared. Most query languages do not provide direct confirmation of what was requested, so that the only validation the user has is the result of the search. The effective use of query languages therefore requires some experience. A specialized example is the web search engine.

3.5.5 Form-fills and spreadsheets

Form-filling interfaces are used primarily for data entry but can also be useful in data retrieval applications. The user is presented with a display resembling a paper



Go-faster Travel Agency Booking

Please enter details of journey:

Start from:	Lancaster
Destination:	Atlanta
Via:	Leeds

[✓ First class] / Second class / Bargain

Single / Return

Seat number:

Figure 3.9 A typical form-filling interface. Screen shot frame reprinted by permission from Microsoft Corporation

form, with slots to fill in (see Figure 3.9). Often the form display is based upon an actual form with which the user is familiar, which makes the interface easier to use. The user works through the form, filling in appropriate values. The data are then entered into the application in the correct place. Most form-filling interfaces allow easy movement around the form and allow some fields to be left blank. They also require correction facilities, as users may change their minds or make a mistake about the value that belongs in each field. The dialog style is useful primarily for data entry applications and, as it is easy to learn and use, for novice users. However, assuming a design that allows flexible entry, form filling is also appropriate for expert users.

Spreadsheets are a sophisticated variation of form filling. The spreadsheet comprises a grid of cells, each of which can contain a value or a formula (see Figure 3.10). The formula can involve the values of other cells (for example, the total of all cells in this column). The user can enter and alter values and formulae in any order and the system will maintain consistency amongst the values displayed, ensuring that all formulae are obeyed. The user can therefore manipulate values to see the effects of changing different parameters. Spreadsheets are an attractive medium for interaction: the user is free to manipulate values at will and the distinction between input and output is blurred, making the interface more flexible and natural.

	Pooches Pet Emporium				
Date	*Description*	*Dog*	*Income*	*Outgoings*	*Balance*
9/2/02	Fees – Mr C. Brown	Snoopy	96.37		96.37
10/2/02	Rubber bones			36.26	60.11
10/2/02	Fees – Mrs E. R. Windsor	7 corgis	1006.45		1066.56
12/2/02	Special order: 7 red carpets			47.28	992.28
16/2/02	Fees – Master T. Tin	Snowy	32.98		1025.26
17/2/02	Beefy Bruno's Bonemeal			243.47	781.79
21/2/02	Fees – Mr F. Flintstone	Dino	21.95		803.74
21/2/02	Special order: 1 Brontosaurus bone			6.47	797.27
28/2/02	Wages – Mr S. H. Ovelit			489.46	307.81

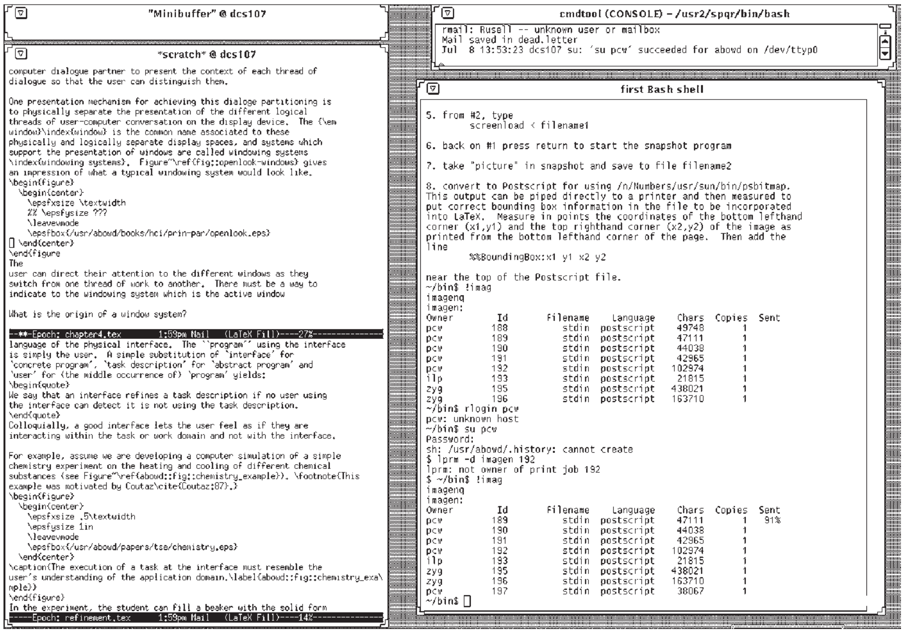
Figure 3.10 A typical spreadsheet

3.5.6 The WIMP interface

Currently many common environments for interactive computing are examples of the *WIMP* interface style, often simply called windowing systems. WIMP stands for windows, icons, menus and pointers (sometimes windows, icons, mice and pull-down menus), and is the default interface style for the majority of interactive computer systems in use today, especially in the PC and desktop workstation arena. Examples of WIMP interfaces include Microsoft Windows for IBM PC compatibles, MacOS for Apple Macintosh compatibles and various X Windows-based systems for UNIX.

Mixing styles

The UNIX windowing environments are interesting as the contents of many of the windows are often themselves simply command line or character-based programs (see Figure 3.11). In fact, this mixing of interface styles in the same system is quite common, especially where older *legacy systems* are used at the same time as more modern applications. It can be a problem if users attempt to use commands and methods suitable for one environment in another. On the Apple Macintosh, HyperCard uses a point-and-click style. However, HyperCard stack buttons look very like Macintosh folders. If you double click on them, as you would to open a folder, your two mouse clicks are treated as separate actions. The first click opens the stack (as you wanted), but the second is then interpreted in the context of the newly opened stack, behaving in an apparently arbitrary fashion! This is an example of the importance of *consistency* in the interface, an issue we shall return to in Chapter 7.



```
computer dialogue partner to present the context of each thread of
dialogue so that the user can distinguish them.

One presentation mechanism for achieving this dialoge partitioning is
to physically separate the presentation of the different logical
threads of user-computer conversation on the display device.  The {\em
window}\index{window} is the common name associated to these
physically and logically separate display spaces, and systems which
support the presentation of windows are called windowing systems
\index{windowing systems}.  Figure~\ref{fig::openlook-windows} gives
an impression of what a typical windowing system would look like.
\begin{figure}
  \begin{center}
    \epsfxsize \textwidth
    %% \epsfysize ???
    \leavevmode
    \epsfbox{/usr/abowd/books/hci/prin-par/openlook.eps}
▯ \end{center}
\end{figure
The
user can direct their attention to the different windows as they
switch from one thread of work to another.  There must be a way to
indicate to the windowing system which is the active window

What is the origin of a window system?

--**-Epoch: chapter4.tex       1:59pm Mail   (LaTeX Fill)----27%-------------
language of the physical interface.  The ``program'' using the interface
is simply the user.  A simple substitution of `interface' for
`concrete program', `task description' for `abstract program' and
`user' for (the middle occurrence of) `program' yields:
\begin{quote}
We say that an interface refines a task description if no user using
the interface can detect it is not using the task description.
\end{quote}
Colloquially, a good interface lets the user feel as if they are
interacting within the task or work domain and not with the interface.

For example, assume we are developing a computer simulation of a simple
chemistry experiment on the heating and cooling of different chemical
substances (see Figure~\ref{abowd::fig::chemistry_example}). \footnote{This
example was motivated by Coutaz\cite{Coutaz:87}.}
\begin{figure}
  \begin{center}
    \epsfxsize .5\textwidth
    \epsfysize 1in
    \leavevmode
    \epsfbox{/usr/abowd/papers/tse/chemistry.eps}
  \end{center}
\caption{The execution of a task at the interface must resemble the
user's understanding of the application domain.\label{abowd::fig::chemistry_exa\
mple}}
\end{figure}
In the experiment, the student can fill a beaker with the solid form
----Epoch: refinement.tex     1:59pm Mail   (LaTeX Fill)----14%-------------
```

```
rmail: Rusell -- unknown user or mailbox
Mail saved in dead.letter
Jul  8 13:53:23 dcs107 su: 'su pcw' succeeded for abowd on /dev/ttyp0
```

```
5. from #2, type
        screenload < filename1

6. back on #1 press return to start the snapshot program

7. take "picture" in snapshot and save to file filename2

8. convert to Postscript for using /n/Numbers/usr/sun/bin/psbitmap.
This output can be piped directly to a printer and then measured to
put correct bounding box information in the file to be incorporated
into LaTeX.  Measure in points the coordinates of the bottom lefthand
corner (x1,y1) and the top righthand corner (x2,y2) of the image as
printed from the bottom lefthand corner of the page.  Then add the
line
        %%BoundingBox:x1 y1 x2 y2

near the top of the Postscript file.
~/bin$ !imag
imagenq
imagen:
Owner        Id       Filename   Language    Chars  Copies  Sent
pcw         188          stdin  postscript    49748       1
pcw         189          stdin  postscript    47111       1
pcw         190          stdin  postscript    44038       1
pcw         191          stdin  postscript    42965       1
pcw         192          stdin  postscript   102974       1
ilp         193          stdin  postscript    21815       1
zyg         195          stdin  postscript   438021       1
zyg         196          stdin  postscript   163710       1
~/bin$ rlogin pcw
pcw: unknown host
~/bin$ su pcw
Password:
sh: /usr/abowd/.history: cannot create
$ lprm -d imagen 192
lprm: not owner of print job 192
$ ~/bin$ !imag
imagenq
imagen:
Owner        Id       Filename   Language    Chars  Copies  Sent
pcw         189          stdin  postscript    47111       1   91%
pcw         190          stdin  postscript    44038       1
pcw         191          stdin  postscript    42965       1
pcw         192          stdin  postscript   102974       1
ilp         193          stdin  postscript    21815       1
zyg         195          stdin  postscript   438021       1
zyg         196          stdin  postscript   163710       1
pcw         197          stdin  postscript    38067       1
~/bin$ ▯
```

Figure 3.11 A typical UNIX windowing system – the OpenLook system. Source: Sun Microsystems, Inc.

3.5.7 Point-and-click interfaces

In most multimedia systems and in web browsers, virtually all actions take only a single click of the mouse button. You may point at a city on a map and when you click a window opens, showing you tourist information about the city. You may point at a word in some text and when you click you see a definition of the word. You may point at a recognizable iconic button and when you click some action is performed.

This point-and-click interface style is obviously closely related to the WIMP style. It clearly overlaps in the use of buttons, but may also include other WIMP elements. However, the philosophy is simpler and more closely tied to ideas of *hypertext*. In addition, the point-and-click style is not tied to mouse-based interfaces, and is also extensively used in touchscreen information systems. In this case, it is often combined with a menu-driven interface.

The point-and-click style has been popularized by world wide web pages, which incorporate all the above types of point-and-click navigation: highlighted words, maps and iconic buttons.

3.5.8 Three-dimensional interfaces

There is an increasing use of three-dimensional effects in user interfaces. The most obvious example is virtual reality, but VR is only part of a range of 3D techniques available to the interface designer.

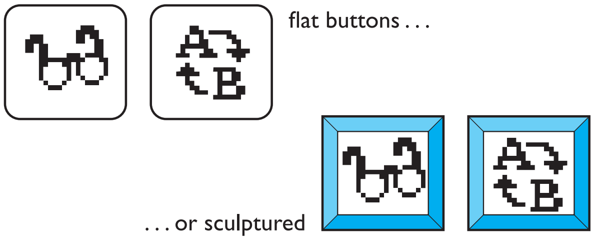


Figure 3.12 Buttons in 3D say 'press me'

The simplest technique is where ordinary WIMP elements, buttons, scroll bars, etc., are given a 3D appearance using shading, giving the appearance of being sculpted out of stone. By unstated convention, such interfaces have a light source at their top right. Where used judiciously, the raised areas are easily identifiable and can be used to highlight active areas (Figure 3.12). Unfortunately, some interfaces make indiscriminate use of sculptural effects, on every text area, border and menu, so all sense of differentiation is lost.

A more complex technique uses interfaces with 3D workspaces. The objects displayed in such systems are usually flat, but are displayed in perspective when at an angle to the viewer and shrink when they are 'further away'. Figure 3.13 shows one such system, WebBook [57]. Notice how size, light and occlusion provide a sense of

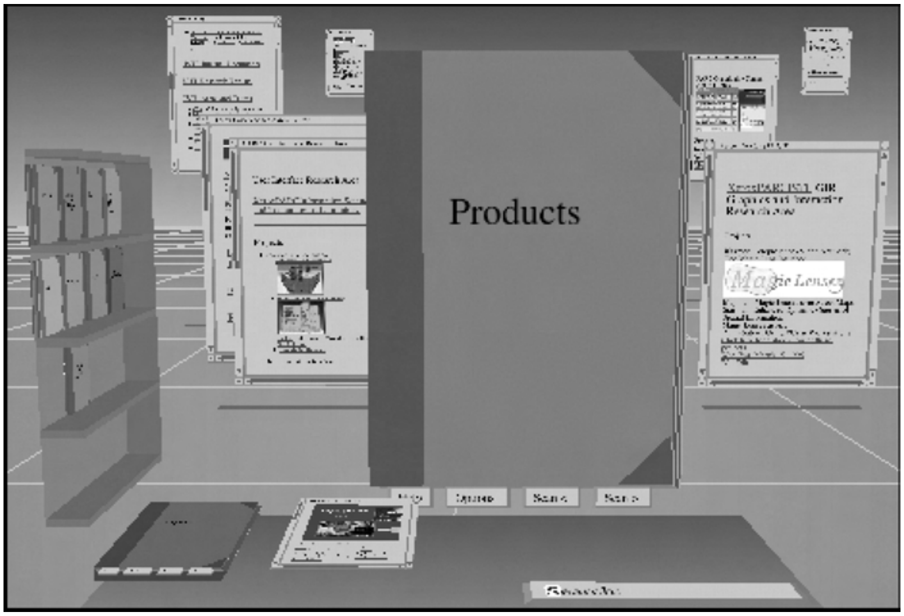


Figure 3.13 WebBook – using 3D to make more space (Card S.K., Robertson G.G. and York W. (1996). The WebBook and the Web Forager: An Information workspace for the World-Wide Web. *CHI96 Conference Proceedings*, 111–17. Copyright © 1996 ACM, Inc. Reprinted by permission)

distance. Notice also that as objects get further away they take up less screen space. Three-dimensional workspaces give you extra space, but in a more natural way than iconizing windows.

Finally, there are virtual reality and information visualization systems where the user can move about within a simulated 3D world. These are discussed in detail in Chapter 20.

These mechanisms overlap with other interaction styles, especially the use of sculptured elements in WIMP interfaces. However, there is a distinct interaction style for 3D interfaces in that they invite us to use our tacit abilities for the real world, and translate them into the electronic world. Novice users must learn that an oval area with a word or picture in it is a button to be pressed, but a 3D button says 'push me'. Further, more complete 3D environments invite one to move within the virtual environment, rather than watch as a spectator.

DESIGN FOCUS

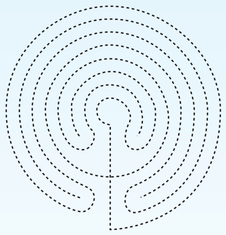
Navigation in 3D and 2D

We live in a three-dimensional world. So clearly 3D interfaces are good . . . or are they? Actually, our 3D stereo vision only works well close to us and after that we rely on cruder measures such as 'this is in front of that'. We are good at moving obects around with our hands in three dimensions, rotating, turning them on their side. However, we walk around in two dimensions and do not fly. Not surprisingly, people find it hard to visualize and control movement in three dimensions.

Normally, we use gravity to give us a fixed direction in space. This is partly through the channels in the inner ear, but also largely through kinesthetic senses – feeling the weight of limbs. When we lose these senses it is easy to become disoriented and we can lose track of which direction is up: divers are trained to watch the direction their bubbles move and if buried in an avalanche you should spit and feel which direction the spittle flows.

Where humans have to navigate in three dimensions they need extra aids such as the artificial horizon in an airplane. Helicopters, where there are many degrees of freedom, are particularly difficult.

Even in the two-dimensional world of walking about we do not rely on neat Cartesian maps in our head. Instead we mostly use models of location such as 'down the road near the church' that rely on approximate topological understanding and landmarks. We also rely on properties of normal space, such as the ability to go backwards and the fact that things that are close can be reached quickly. When two-dimensional worlds are not like this, for example in a one-way traffic system or in a labyrinth, we have great difficulty [98].



When we design systems we should take into account how people navigate in the real world and use this to guide our navigation aids. For example, if we have a 3D interface or a virtual reality world we should normally show a ground plane and by default lock movement to be parallel to the ground. In information systems we can recruit our more network-based models of 2D space by giving landmarks and making it as easy to 'step back' as to go forwards (as with the web browser 'back' button).

See the book website for more about 3D vision: /e3/online/seeing-3D/

3.6 ELEMENTS OF THE WIMP INTERFACE

We have already noted the four key features of the WIMP interface that give it its name – windows, icons, pointers and menus – and we will now describe these in turn. There are also many additional interaction objects and techniques commonly used in WIMP interfaces, some designed for specific purposes and others more general. We will look at buttons, toolbars, palettes and dialog boxes. Most of these elements can be seen in Figure 3.14.

Together, these elements of the WIMP interfaces are called *widgets*, and they comprise the toolkit for interaction between user and system. In Chapter 8 we will describe windowing systems and interaction widgets more from the programmer's perspective. There we will discover that though most modern windowing systems provide the same set of basic widgets, the 'look and feel' – how widgets are physically displayed and how users can interact with them to access their functionality – of different windowing systems and toolkits can differ drastically.

3.6.1 Windows

Windows are areas of the screen that behave as if they were independent terminals in their own right. A window can usually contain text or graphics, and can be moved

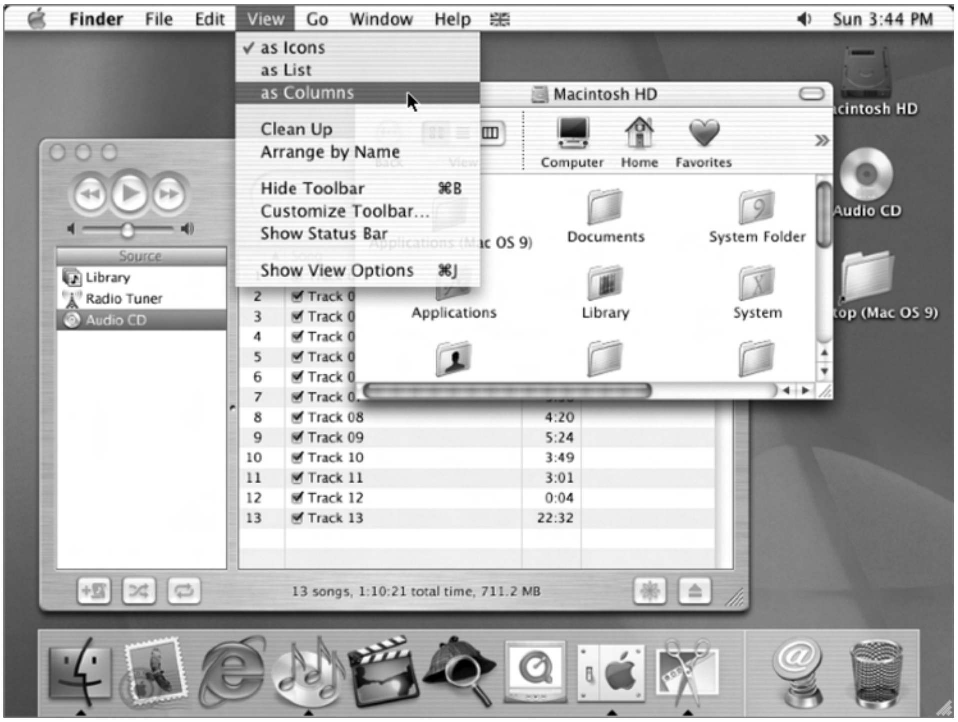


Figure 3.14 Elements of the WIMP interface – Microsoft Word 5.1 on an Apple Macintosh. Screen shot reprinted by permission from Apple Computer, Inc.

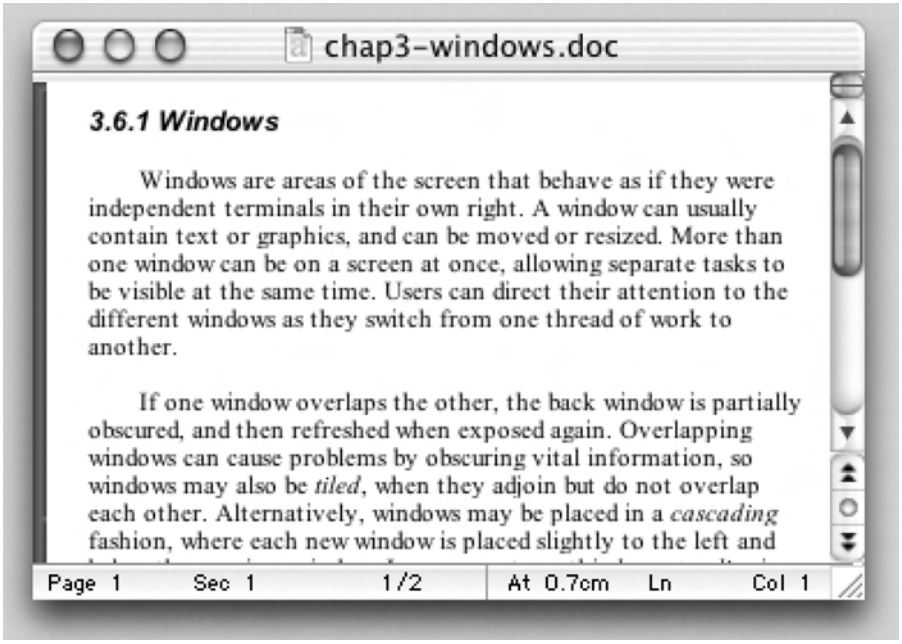
or resized. More than one window can be on a screen at once, allowing separate tasks to be visible at the same time. Users can direct their attention to the different windows as they switch from one thread of work to another.

If one window overlaps the other, the back window is partially obscured, and then refreshed when exposed again. Overlapping windows can cause problems by obscuring vital information, so windows may also be *tiled*, when they adjoin but do not overlap each other. Alternatively, windows may be placed in a *cascading* fashion, where each new window is placed slightly to the left and below the previous window. In some systems this *layout policy* is fixed, in others it can be selected by the user.

Usually, windows have various things associated with them that increase their usefulness. *Scrollbars* are one such attachment, allowing the user to move the contents of the window up and down, or from side to side. This makes the window behave as if it were a real window onto a much larger world, where new information is brought into view by manipulating the scrollbars.

There is usually a title bar attached to the top of a window, identifying it to the user, and there may be special boxes in the corners of the window to aid resizing, closing, or making as large as possible. Each of these can be seen in Figure 3.15.

In addition, some systems allow windows within windows. For example, in Microsoft Office applications, such as Excel and Word, each application has its own window and then within this each document has a window. It is often possible to have different layout policies within the different application windows.


chap3-windows.doc

3.6.1 Windows

Windows are areas of the screen that behave as if they were independent terminals in their own right. A window can usually contain text or graphics, and can be moved or resized. More than one window can be on a screen at once, allowing separate tasks to be visible at the same time. Users can direct their attention to the different windows as they switch from one thread of work to another.

If one window overlaps the other, the back window is partially obscured, and then refreshed when exposed again. Overlapping windows can cause problems by obscuring vital information, so windows may also be *tiled*, when they adjoin but do not overlap each other. Alternatively, windows may be placed in a *cascading* fashion, where each new window is placed slightly to the left and

Page 1 Sec 1 1/2 At 0.7cm Ln Col 1

Figure 3.15 A typical window. Screen shot reprinted by permission from Apple Computer, Inc.

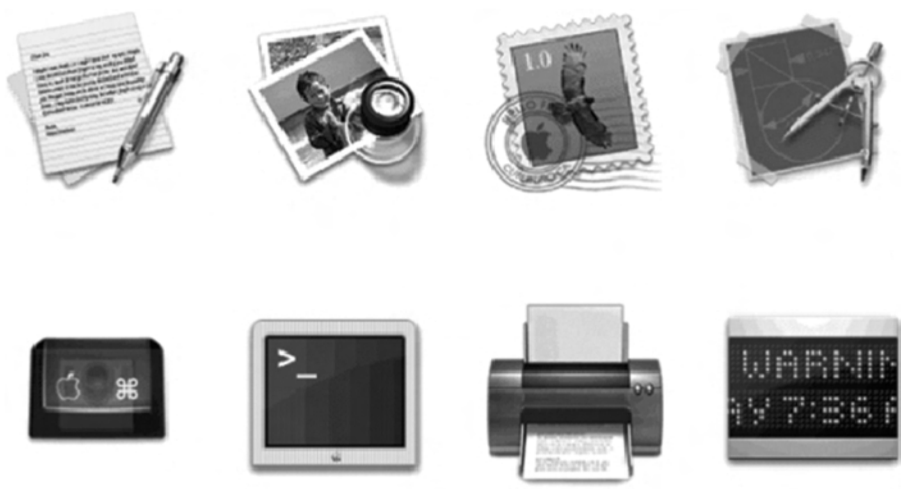


Figure 3.16 A variety of icons. Screen shot reprinted by permission from Apple Computer, Inc.

3.6.2 Icons

Windows can be closed and lost for ever, or they can be shrunk to some very reduced representation. A small picture is used to represent a closed window, and this representation is known as an *icon*. By allowing icons, many windows can be available on the screen at the same time, ready to be expanded to their full size by clicking on the icon. Shrinking a window to its icon is known as *iconifying* the window. When a user temporarily does not want to follow a particular thread of dialog, he can suspend that dialog by iconifying the window containing the dialog. The icon saves space on the screen and serves as a reminder to the user that he can subsequently resume the dialog by opening up the window. Figure 3.16 shows a few examples of some icons used in a typical windowing system (MacOS X).

Icons can also be used to represent other aspects of the system, such as a wastebasket for throwing unwanted files into, or various disks, programs or functions that are accessible to the user. Icons can take many forms: they can be realistic representations of the objects that they stand for, or they can be highly stylized. They can even be arbitrary symbols, but these can be difficult for users to interpret.

3.6.3 Pointers

The pointer is an important component of the WIMP interface, since the interaction style required by WIMP relies very much on pointing and selecting things such as icons. The mouse provides an input device capable of such tasks, although joysticks and trackballs are other alternatives, as we have previously seen in Chapter 2. The user is presented with a cursor on the screen that is controlled by the input device. A variety of pointer cursors are shown in Figure 3.17.

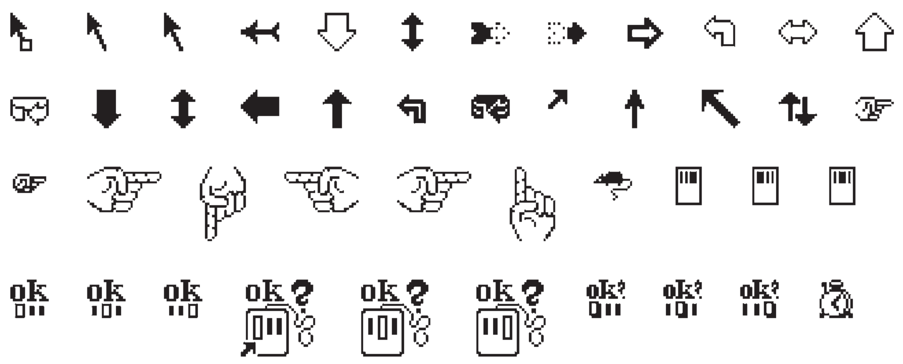


Figure 3.17 A variety of pointer cursors. Source: Sun Microsystems, Inc.

The different shapes of cursor are often used to distinguish *modes*, for example the normal pointer cursor may be an arrow, but change to cross-hairs when drawing a line. Cursors are also used to tell the user about system activity, for example a watch or hour-glass cursor may be displayed when the system is busy reading a file.

Pointer cursors are like icons, being small bitmap images, but in addition all cursors have a *hot-spot*, the location to which they point. For example, the three arrows at the start of Figure 3.17 each have a hot-spot at the top left, whereas the right-pointing hand on the second line has a hot-spot on its right. Sometimes the hot-spot is not clear from the appearance of the cursor, in which case users will find it hard to click on small targets. When designing your own cursors, make sure the image has an obvious hot-spot.

3.6.4 Menus

The last main feature of windowing systems is the *menu*, an interaction technique that is common across many non-windowing systems as well. A menu presents a choice of operations or services that can be performed by the system at a given time. In Chapter 1, we pointed out that our ability to recall information is inferior to our ability to recognize it from some visual cue. Menus provide information cues in the form of an ordered list of operations that can be scanned. This implies that the names used for the commands in the menu should be meaningful and informative.

The pointing device is used to indicate the desired option. As the pointer moves to the position of a menu item, the item is usually highlighted (by inverse video, or some similar strategy) to indicate that it is the potential candidate for selection. Selection usually requires some additional user action, such as pressing a button on the mouse that controls the pointer cursor on the screen or pressing some special key on the keyboard. Menus are inefficient when they have too many items, and so cascading menus are utilized, in which item selection opens up another menu adjacent to the item, allowing refinement of the selection. Several layers of cascading menus can be used.

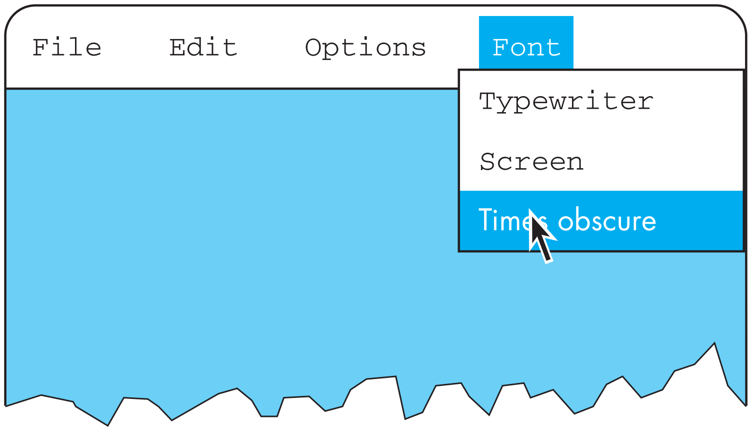


Figure 3.18 Pull-down menu

The main menu can be visible to the user all the time, as a *menu bar* and submenus can be pulled down or across from it upon request (Figure 3.18). Menu bars are often placed at the top of the screen (for example, MacOS) or at the top of each window (for example, Microsoft Windows). Alternatives include menu bars along one side of the screen, or even placed amongst the windows in the main 'desktop' area. Websites use a variety of menu bar locations, including top, bottom and either side of the screen. Alternatively, the main menu can be hidden and upon request it will pop up onto the screen. These *pop-up menus* are often used to present context-sensitive options, for example allowing one to examine properties of particular on-screen objects. In some systems they are also used to access more global actions when the mouse is depressed over the screen background.

Pull-down menus are dragged down from the title at the top of the screen, by moving the mouse pointer into the title bar area and pressing the button. Fall-down menus are similar, except that the menu automatically appears when the mouse pointer enters the title bar, without the user having to press the button. Some menus are pin-up menus, in that they can be 'pinned' to the screen, staying in place until explicitly asked to go away. Pop-up menus appear when a particular region of the screen, maybe designated by an icon, is selected, but they only stay as long as the mouse button is depressed.

Another approach to menu selection is to arrange the options in a circular fashion. The pointer appears in the center of the circle, and so there is the same distance to travel to any of the selections. This has the advantages that it is easier to select items, since they can each have a larger target area, and that the selection time for each item is the same, since the pointer is equidistant from them all. Compare this with a standard menu: remembering Fitts' law from Chapter 1, we can see that it will take longer to select items near the bottom of the menu than at the top. However, these *pie menus*, as they are known [54], take up more screen space and are therefore less common in interfaces.

Keyboard accelerators

Menus often offer *keyboard accelerators*, key combinations that have the same effect as selecting the menu item. This allows more expert users, familiar with the system, to manipulate things without moving off the keyboard, which is often faster. The accelerators are often displayed alongside the menu items so that frequent use makes them familiar. Unfortunately most systems do not allow you to use the accelerators while the menu is displayed. So, for example, the menu might say

```
Find          F3
```

However, when the user presses function key F3 nothing happens. F3 only works when the menu is *not* displayed – when the menu is there you must press 'F' instead! This is an example of an interface that is *dishonest* (see also Chapter 7).

The major problems with menus in general are deciding what items to include and how to group those items. Including too many items makes menus too long or creates too many of them, whereas grouping causes problems in that items that relate to the same topic need to come under the same heading, yet many items could be grouped under more than one heading. In pull-down menus the menu label should be chosen to reflect the function of the menu items, and items grouped within menus by function. These groupings should be consistent across applications so that the user can transfer learning to new applications. Menu items should be ordered in the menu according to importance and frequency of use, and opposite functionalities (such as 'save' and 'delete') should be kept apart to prevent accidental selection of the wrong function, with potentially disastrous consequences.

3.6.5 Buttons

Buttons are individual and isolated regions within a display that can be selected by the user to invoke specific operations. These regions are referred to as buttons because they are purposely made to resemble the push buttons you would find on a control panel. 'Pushing' the button invokes a command, the meaning of which is usually indicated by a textual label or a small icon. Buttons can also be used to toggle between two states, displaying status information such as whether the current font is italicized or not in a word processor, or selecting options on a web form. Such toggle buttons can be grouped together to allow a user to select one feature from a set of mutually exclusive options, such as the size in points of the current font. These are called *radio buttons*, since the collection functions much like the old-fashioned mechanical control buttons on car radios. If a set of options is not mutually exclusive, such as font characteristics like bold, italics and underlining, then a set of toggle buttons can be used to indicate the on/off status of the options. This type of collection of buttons is sometimes referred to as *check boxes*.

3.6.6 Toolbars

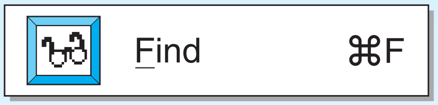
Many systems have a collection of small buttons, each with icons, placed at the top or side of the window and offering commonly used functions. The function of this *toolbar* is similar to a menu bar, but as the icons are smaller than the equivalent text more functions can be simultaneously displayed. Sometimes the content of the toolbar is fixed, but often users can *customize* it, either changing which functions are made available, or choosing which of several predefined toolbars is displayed.

Design Focus

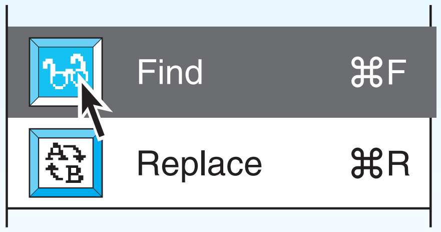
Learning toolbars

Although many applications now have toolbars, they are often underused because users simply do not know what the icons represent. Once learned the meaning is often relatively easy to remember, but most users do not want to spend time reading a manual, or even using online help to find out what each button does – they simply reach for the menu.

There is an obvious solution – put the icons on the menus in the same way that accelerator keys are written there. So in the 'Edit' menu one might find the option



Imagine now selecting this. As the mouse drags down through the menu selections, each highlights in turn. If the mouse is dragged down the extreme left, the effect will be very similar to selecting the icon from the toolbar, except that it will be incidental to selecting the menu item. In this way, the toolbar icon will be naturally learned from normal menu interaction.



Selecting the menu option = selecting the icon

This trivial fix is based on accepted and tested knowledge of learning and has been described in more detail by one of the authors elsewhere [95]. Given its simplicity, this technique should clearly be used everywhere, but until recently was rare. However, it has now been taken up in the Office 97 suite and later Microsoft Office products, so perhaps will soon become standard.

3.6.7 Palettes

In many application programs, interaction can enter one of several *modes*. The defining characteristic of modes is that the interpretation of actions, such as keystrokes or gestures with the mouse, changes as the mode changes. For example, using the standard UNIX text editor vi, keystrokes can be interpreted either as operations to insert characters in the document (insert mode) or as operations to perform file manipulation (command mode). Problems occur if the user is not aware of the current mode. Palettes are a mechanism for making the set of possible modes and the active mode visible to the user. A palette is usually a collection of icons that are reminiscent of the purpose of the various modes. An example in a drawing package would be a collection of icons to indicate the pixel color or pattern that is used to fill in objects, much like an artist's palette for paint.

Some systems allow the user to create palettes from menus or toolbars. In the case of pull-down menus, the user may be able 'tear off' the menu, turning it into a palette showing the menu items. In the case of toolbars, he may be able to drag the toolbar away from its normal position and place it anywhere on the screen. Tear-off menus are usually those that are heavily graphical anyway, for example line-style or color selection in a drawing package.

3.6.8 Dialog boxes

Dialog boxes are information windows used by the system to bring the user's attention to some important information, possibly an error or a warning used to prevent a possible error. Alternatively, they are used to invoke a subdialog between user and system for a very specific task that will normally be embedded within some larger task. For example, most interactive applications result in the user creating some file that will have to be named and stored within the filing system. When the user or system wants to save the file, a dialog box can be used to allow the user to name the file and indicate where it is to be located within the filing system. When the save subdialog is complete, the dialog box will disappear. Just as windows are used to separate the different threads of user–system dialog, so too are dialog boxes used to factor out auxiliary task threads from the main task dialog.

3.7 INTERACTIVITY

When looking at an interface, it is easy to focus on the visually distinct parts (the buttons, menus, text areas) but the dynamics, the way they react to a user's actions, are less obvious. Dialog design, discussed in Chapter 16, is focussed almost entirely on the choice and specification of appropriate sequences of actions and corresponding changes in the interface state. However, it is typically not used at a fine level of detail and deliberately ignores the 'semantic' level of an interface: for example, the validation of numeric information in a forms-based system.

It is worth remembering that *interactivity* is the defining feature of an *interactive* system. This can be seen in many areas of HCI. For example, the recognition rate for *speech recognition* is too low to allow transcription from tape, but in an airline reservation system, so long as the system can reliably recognize *yes* and *no* it can reflect back its understanding of what you said and seek confirmation. Speech-based *input* is difficult, speech-based *interaction* easier. Also, in the area of information visualization the most exciting developments are all where users can interact with a visualization in real time, changing parameters and seeing the effect.

Interactivity is also crucial in determining the 'feel' of a WIMP environment. All WIMP systems appear to have virtually the same elements: windows, icons, menus, pointers, dialog boxes, buttons, etc. However, the precise behavior of these elements differs both within a single environment and between environments. For example, we have already discussed the different behavior of pull-down and fall-down menus. These look the same, but fall-down menus are more easily invoked by accident (and not surprisingly the windowing environments that use them have largely fallen into disuse!). In fact, menus are a major difference between the MacOS and Microsoft Windows environments: in MacOS you have to keep the mouse depressed throughout menu selection; in Windows you can click on the menu bar and a pull-down menu appears and remains there until an item is selected or it is cancelled. Similarly the detailed behavior of buttons is quite complex, as we shall see in Chapter 17.

In older computer systems, the order of interaction was largely determined by the machine. You did things when the computer was ready. In WIMP environments, the user takes the initiative, with many options and often many applications simultaneously available. The exceptions to this are *pre-emptive* parts of the interface, where the system for various reasons wrests the initiative away from the user, perhaps because of a problem or because it needs information in order to continue.

The major example of this is *modal dialog boxes*. It is often the case that when a dialog box appears the application will not allow you to do anything else until the dialog box has been completed or cancelled. In some cases this may simply block the application, but you can perform tasks in other applications. In other cases you can do nothing at all until the dialog box has been completed. An especially annoying example is when the dialog box asks a question, perhaps simply for confirmation of an action, but the information you need to answer is hidden by the dialog box!

There are occasions when modal dialog boxes are necessary, for example when a major fault has been detected, or for certain kinds of instructional software. However, the general philosophy of modern systems suggests that one should minimize the use of pre-emptive elements, allowing the user maximum flexibility.

Interactivity is also critical in dealing with errors. We discussed slips and mistakes earlier in the chapter, and some ways to try to prevent these types of errors. The other way to deal with errors is to make sure that the user or the system is able to tell when errors have occurred. If users can *detect* errors then they can correct them. So, even if errors occur, the interaction as a whole succeeds. Several of the principles in Chapter 7 deal with issues that relate to this. This ability to detect and correct is important both at the small scale of button presses and keystrokes and also at the large scale. For example, if you have sent a client a letter and expect a reply, you can put in your diary a note on the day you expect a reply. If the other person forgets to reply or the letter gets lost in the post you know to send a reminder or ring when the due day passes.

3.8 THE CONTEXT OF THE INTERACTION

We have been considering the interaction between a user and a system, and how this is affected by interface design. This interaction does not occur within a vacuum. We have already noted some of the physical factors in the environment that can directly affect the quality of the interaction. This is part of the context in which the interaction takes place. But this still assumes a single user operating a single, albeit complex, machine. In reality, users work within a wider social and organizational context. This provides the wider context for the interaction, and may influence the activity and motivation of the user. In Chapter 13, we discuss some methods that can be used to gain a fuller understanding of this context, and, in Chapter 14, we consider in more detail the issues involved when more than one user attempts to work together on a system. Here we will confine our discussion to the influence social and organizational factors may have on the user's interaction with the system. These may not be factors over which the designer has control. However, it is important to be aware of such influences to understand the user and the work domain fully.

Bank managers don't type . . .

The safe in most banks is operated by at least two keys, held by different employees of the bank. This makes it difficult for a bank robber to obtain both keys, and also protects the bank against light-fingered managers! ATMs contain a lot of cash and so need to be protected by similar measures. In one bank, which shall remain nameless, the ATM had an electronic locking device. The machine could not be opened to replenish or remove cash until a long key sequence had been entered. In order to preserve security, the bank gave half the sequence to one manager and half to another, so both managers had to be present in order to open the ATM. However, these were traditional bank managers who were not used to typing – that was a job for a secretary! So they each gave their part of the key sequence to a secretary to type in when they wanted to gain entry to the ATM. In fact, they both gave their respective parts of the key sequence to the *same* secretary. Happily the secretary was honest, but the moral is you cannot ignore social expectations and relationships when designing any sort of computer system, however simple it may be.

The presence of other people in a work environment affects the performance of the worker in any task. In the case of peers, competition increases performance, at least for known tasks. Similarly the desire to impress management and superiors improves performance on these tasks. However, when it comes to acquisition of new skills, the presence of these groups can inhibit performance, owing to the fear of failure. Consequently, privacy is important to allow users the opportunity to experiment.

In order to perform well, users must be motivated. There are a number of possible sources of motivation, as well as those we have already mentioned, including fear, allegiance, ambition and self-satisfaction. The last of these is influenced by the user's perception of the quality of the work done, which leads to job satisfaction. If a system makes it difficult for the user to perform necessary tasks, or is frustrating to use, the user's job satisfaction, and consequently performance, will be reduced.

The user may also lose motivation if a system is introduced that does not match the actual requirements of the job to be done. Often systems are chosen and introduced by managers rather than the users themselves. In some cases the manager's perception of the job may be based upon observation of results and not on actual activity. The system introduced may therefore impose a way of working that is unsatisfactory to the users. If this happens there may be three results: the system will be rejected, the users will be resentful and unmotivated, or the user will adapt the intended interaction to his own requirements. This indicates the importance of involving actual users in the design process.

Design Focus

Half the picture?

When systems are not designed to match the way people actually work, then users end up having to do 'work arounds'. Integrated student records systems are becoming popular in universities in the UK. They bring the benefits of integrating examination systems with enrolment and finance systems so all data can be maintained together and cross-checked. All very useful and time saving – in theory. However, one commonly used system only holds a single overall mark per module for each student, whereas many modules on UK courses have multiple elements of assessment. Knowing a student's mark on each part of the assessment is often useful to academics making decisions in examination boards as it provides a more detailed picture of performance. In many cases staff are therefore supplementing the official records system with their own unofficial spreadsheets to provide this information – making additional work for staff and increased opportunity for error.

On the other hand, the introduction of new technology may prove to be a motivation to users, particularly if it is well designed, integrated with the user's current work, and challenging. Providing adequate feedback is an important source of motivation for users. If no feedback is given during a session, the user may become bored, unmotivated or, worse, unsure of whether the actions performed have been successful. In general, an action should have an obvious effect to prevent this confusion and to allow early recovery in the case of error. Similarly, if system delays occur, feedback can be used to prevent frustration on the part of the user – the user is then aware of what is happening and is not left wondering if the system is still working.

3.9 EXPERIENCE, ENGAGEMENT AND FUN

Ask many in HCI about usability and they may use the words 'effective' and 'efficient'. Some may add 'satisfaction' as well. This view of usability seems to stem mainly from the Taylorist tradition of time and motion studies: if you can get the worker to pull the levers and turn the knobs in the right order then you can shave 10% off production costs.

However, users no longer see themselves as cogs in a machine. Increasingly, applications are focussed outside the closed work environment: on the home, leisure, entertainment, shopping. It is not sufficient that people can use a system, they must *want* to use it.

Even from a pure economic standpoint, your employees are likely to work better and more effectively if they enjoy what they are doing!

In this section we'll look at these more experiential aspects of interaction.

3.9.1 Understanding experience

Shopping is an interesting example to consider. Most internet stores allow you to buy things, but do you go shopping? Shopping is as much about going to the shops, feeling the clothes, being with friends. You can go shopping and never intend to spend money. Shopping is not about an efficient financial transaction, it is an experience.

But experience is a difficult thing to pin down; we understand the idea of a good experience, but how do we define it and even more difficult how do we design it?

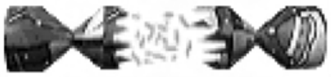
Csikszentimihalyi [82] looked at extreme experiences such as climbing a rock face in order to understand that feeling of total engagement that can sometimes happen. He calls this *flow* and it is perhaps related to what some sportspeople refer to as being 'in the zone'. This sense of flow occurs when there is a balance between anxiety and boredom. If you do something that you know you can do it is not engaging; you may do it automatically while thinking of something else, or you may simply become bored. Alternatively, if you do something completely outside your abilities you may become anxious and, if you are half way up a rock face, afraid. Flow comes when you are teetering at the edge of your abilities, stretching yourself to or a little beyond your limits.

In education there is a similar phenomenon. The *zone of proximal development* is those things that you cannot quite do yourself, but you can do with some support, whether from teachers, fellow pupils, or electronic or physical materials. Learning is at its best in this zone. Notice again this touching of limits.

Of course, this does not fully capture the sense of experience, and there is an active subfield of HCI researchers striving to make sense of this, building on the work of psychologists and philosophers on the one hand and literary analysis, film making and drama on the other.

3.9.2 Designing experience

Some of the authors were involved in the design of virtual Christmas crackers. These are rather like electronic greetings cards, but are based on crackers. For those who have not come across them, Christmas crackers are small tubes of paper between 8 and 12 inches long (20–30 cm). Inside there are a small toy, a joke or motto and a paper hat. A small strip of card is threaded through, partly coated with gunpowder. When two people at a party pull the cracker, it bursts apart with a small bang from the gunpowder and the contents spill out.



The virtual cracker does not attempt to fully replicate each aspect of the physical characteristics and process of pulling the cracker, but instead seeks to reproduce the experience. To do this the original crackers experience was deconstructed and each aspect of the experience produced in a similar, but sometimes different, way in the new media. Table 3.1 shows the aspects of the experience deconstructed and reconstructed in the virtual cracker.

For example, the cracker contents are hidden inside; no one knows what toy or joke will be inside. Similarly, when you create a virtual cracker you normally cannot see the contents until the recipient has opened it. Even the recipient initially sees a page with just an image of the cracker; it is only after the recipient has clicked on the 'open' icon that the cracker slowly opens and you get to see the joke, web toy and mask.

The mask is also worth looking at. The first potential design was to have a picture of a face with a hat on it – well, it wouldn't rank highly on excitement! The essential feature of the paper hat is that you can dress up. An iconic hat hardly does that.

Table 3.1 The crackers experience [101]

	Real cracker	Virtual cracker
Surface elements		
Design	Cheap and cheerful	Simple page/graphics
Play	Plastic toy and joke	Web toy and joke
Dressing up	Paper hat	Mask to cut out
Experienced effects		
Shared	Offered to another	Sent by email, message
Co-experience	Pulled together	Sender can't see content until opened by recipient
Excitement	Cultural connotations	Recruited expectation
Hiddenness	Contents inside	First page – no contents
Suspense	Pulling cracker	Slow . . . page change
Surprise	Bang (when it works)	WAV file (when it works)

Instead the cracker has a link to a web page with a picture of a mask that you can print, cut out and wear. Even if you don't actually print it out, the fact that you could changes the experience – it is some dressing up you just happen not to have done yet.

A full description of the virtual crackers case study is on the book website at: /e3/casestudy/crackers/

3.9.3 Physical design and engagement

In Chapter 2 we talked about physical controls. Figure 2.13 showed controllers for a microwave, washing machine and personal MiniDisc player. We saw then how certain physical interfaces were suited for different contexts: smooth plastic controls for an easy clean microwave, multi-function knob for the MiniDisc.

Designers are faced with many constraints:

Ergonomic You cannot physically push buttons if they are too small or too close.

Physical The size or nature of the device may force certain positions or styles of control, for example, a dial like the one on the washing machine would not fit on the MiniDisc controller; high-voltage switches cannot be as small as low-voltage ones.

Legal and safety Cooker controls must be far enough from the pans that you do not burn yourself, but also high enough to prevent small children turning them on.

Context and environment The microwave's controls are smooth to make them easy to clean in the kitchen.

Aesthetic The controls must look good.

Economic It must not cost too much!

These constraints are themselves often contradictory and require trade-offs to be made. For example, even within the safety category front-mounted controls are better in that they can be turned on or off without putting your hands over the pans and hot steam, but back-mounted controls are further from children's grasp. The MiniDisc player is another example; it physically needs to be small, but this means there is not room for all the controls you want given the minimum size that can be manipulated. In the case of the cooker there is no obvious best solution and so different designs favor one or the other. In the case of the MiniDisc player the end knob is multi-function. This means the knob is ergonomically big enough to turn and physically small enough to fit, but at the cost of a more complex interaction style.

To add to this list of constraints there is another that makes a major impact on the ease of use and also the ability of the user to become engaged with the device, for it to become natural to use:

Fluidity The extent to which the physical structure and manipulation of the device naturally relate to the logical functions it supports.

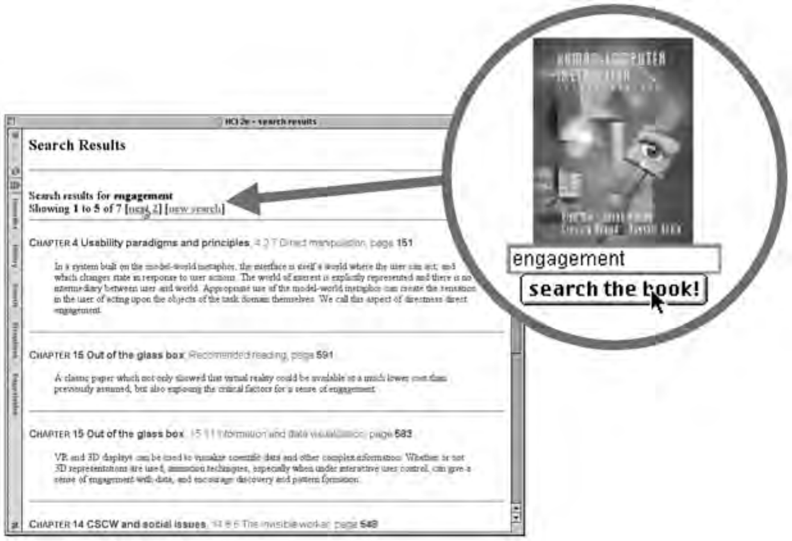
This is related closely to the idea of *affordances*, which we discuss in Section 5.7.2. The knob at the end of the MiniDisc controller affords turning – it is an obvious thing to do. However, this may not have mapped naturally onto the logical functions. Two of the press buttons are for cycling round the display options and for changing sound options. Imagine a design where turning the knob to clockwise cycled through the display options and turning it anti-clockwise cycled through the sound options. This would be a compact design satisfying all the ergonomic, physical and aesthetic constraints, but would not have led to as fluid an interaction. The physically opposite motions lead to logically distinct effects. However, the designers did a better job than this! The twist knob is used to move backwards and forwards through the tracks of the MiniDisc – that is, opposite physical movements produce opposite logical effects. Holding the knob out and twisting turns the volume up and down. Again, although the pull action is not a natural mapping, the twist maps very naturally onto controlling the sound level.

As well as being fluid in action, some controls portray by their physical appearance the underlying state they control. For example, the dial on the washing machine both sets the program and reflects the current stage in the washing cycle as it turns. A simple on/off switch also does this. However, it is also common to see the power on computers and hifi devices controlled by a push button – press for on, then press again for off. The button does not reflect the state at all. When the screen is on this is not a problem as the fact that there is something on the screen acts as a very immediate indicator of the state. But if the screen has a power save then you might accidentally turn the machine off thinking that you are turning it on! For this reason, this type of power button often has a light beside it to show you the power is on. A simple switch tells you that itself!

3.9.4 Managing value

If we want people to *want* to use a device or application we need to understand their personal values. Why should they want to use it? What value do they get from using it? Now when we say value here we don't mean monetary value, although that may be part of the story, but all the things that drive a person. For some people this may include being nice to colleagues, being ecologically friendly, being successful in their career. Whatever their personal values are, if we ask someone to do something or use something they are only likely to do it if the value to them exceeds the cost.

This is complicated by the fact that for many systems the costs such as purchase cost, download time of a free application, learning effort are incurred up front, whereas often the returns – faster work, enjoyment of use – are seen later. In economics, businesses use a measure called 'net present value' to calculate what a future gain is worth today; because money can be invested, £100 today is worth the same as perhaps £200 in five years' time. Future gain is discounted. For human decision making, future gains are typically discounted very highly; many of us are bad at saving for tomorrow or even keeping the best bits of our dinner until last. This means that not only must we understand people's value systems, but we must be able to offer


HCI 2e - search results

Search Results

Search results for **engagement**
Showing 1 to 5 of 7 [next 2] [new search]

CHAPTER 4 Usability paradigms and principles 4.2.7 Direct manipulation, page 151

In a system built on the model-world metaphor, the interface is itself a world where the user can act, and which changes state in response to user actions. The world of interest is explicitly represented and there is no intermediary between user and world. Appropriate use of the model-world metaphor can create the sensation in the user of acting upon the objects of the task domain themselves. We call this aspect of directness direct engagement.

CHAPTER 15 Out of the glass box Recommended reading, page 591

A classic paper which not only showed that virtual reality could be available at a much lower cost than previously assumed, but also exposing the critical factors for a sense of engagement.

CHAPTER 15 Out of the glass box 15.11 Information and data visualization, page 583

VR and 3D displays can be used to visualize scientific data and other complex information. Whether or not 3D representations are used, animation techniques, especially when under interactive user control, can give a sense of engagement with data, and encourage discovery and pattern formation.

CHAPTER 14 CSCW and social issues 14.8.6 The invisible worker, page 549

Figure 3.19 The web-based book search facility. Screen shot frame reprinted by permission from Microsoft Corporation

gains sooner as well as later, or at least produce a very good demonstration of potential future gains so that they have a *perceived* current value.

When we were preparing the website for the second edition of this book we thought very hard about how to give things that were of value to those who had the book, and also to those who hadn't. The latter is partly because we are all academics and researchers in the field and so want to contribute to the HCI community, but also of course we would like lots of people to buy the book. One option we thought of was to put the text online, which would be good for people without the book, but this would have less value to people who have the book (they might even be annoyed that those who hadn't paid should have access). The search mechanism was the result of this process (Figure 3.19). It gives value to those who have the book because it is a way of finding things. It is of value to those who don't because it acts as a sort of online encyclopedia of HCI. However, because it always gives the chapter and page number in the book it also says to those who haven't got the book: 'buy me'. See an extended case study about the design of the book search on the website at /e3/casestudy/search/

3.10 SUMMARY

In this chapter, we have looked at the interaction between human and computer, and, in particular, how we can ensure that the interaction is effective to allow the user to get the required job done. We have seen how we can use Norman's execution–evaluation model, and the interaction framework that extends it, to analyze the interaction in terms of how easy or difficult it is for the user to express what he wants and determine whether it has been done.

We have also looked at the role of ergonomics in interface design, in analyzing the physical characteristics of the interaction, and we have discussed a number of interface styles. We have considered how each of these factors can influence the effectiveness of the interaction.

Interactivity is at the heart of all modern interfaces and is important at many levels. Interaction between user and computer does not take place in a vacuum, but is affected by numerous social and organizational factors. These may be beyond the designer's control, but awareness of them can help to limit any negative effects on the interaction.

EXERCISES

3.1 Choose two of the interface styles (described in Section 3.5) that you have experience of using. Use the interaction framework to analyze the interaction involved in using these interface styles for a database selection task. Which of the distances is greatest in each case?

3.2 Find out all you can about natural language interfaces. Are there any successful systems? For what applications are these most appropriate?

3.3 What influence does the social environment in which you work have on your interaction with the computer? What effect does the organization (commercial or academic) to which you belong have on the interaction?

3.4 (a) Group the following functions under appropriate headings, assuming that they are to form the basis for a menu-driven word-processing system – the headings you choose will become the menu titles, with the functions appearing under the appropriate one. You can choose as many or as few menu headings as you wish. You may also alter the wordings of the functions slightly if you wish.

save, save as, new, delete, open mail, send mail, quit, undo, table, glossary, preferences, character style, format paragraph, lay out document, position on page, plain text, bold text, italic text, underline, open file, close file, open copy of file, increase point size, decrease point size, change font, add footnote, cut, copy, paste, clear, repaginate, add page break, insert graphic, insert index entry, print, print preview, page setup, view page, find word, change word, go to, go back, check spelling, view index, see table of contents, count words, renumber pages, repeat edit, show alternative document, help

(b) If possible, show someone else your headings, and ask them to group the functions under your headings. Compare their groupings with yours. You should find that there are areas of great similarity, and some differences. Discuss the similarities and discrepancies.

Why do some functions always seem to be grouped together?
Why do some groups of functions always get categorized correctly?
Why are some less easy to place under the 'correct' heading?
Why is this important?

3.5 Using your function groupings from Exercise 3.4, count the number of items in your menus.

(a) What is the average?
What is the disadvantage of putting all the functions on the screen at once?
What is the problem with using lots of menu headings?
What is the problem of using very few menu headings?

Consider the following: I can group my functions either into three menus, with lots of functions in each one, or into eight menus with fewer in each. Which will be easier to use? Why?

(b) *Optional experiment*
Design an experiment to test your answers. Perform the experiment and report on your results.

3.6 Describe (in words as well as graphically) the interaction framework introduced in *Human–Computer Interaction*. Show how it can be used to explain problems in the dialog between a user and a computer.

3.7 Describe briefly four different interaction styles used to accommodate the dialog between user and computer.

3.8 The typical computer screen has a WIMP setup (what does WIMP stand for?). Most common WIMP arrangements work on the basis of a desktop metaphor, in which common actions are likened to similar actions in the real world. For example, moving a file is achieved by selecting it and dragging it into a relevant folder or filing cabinet. The advantage of using a metaphor is that the user can identify with the environment presented on the screen. Having a metaphor allows users to predict the outcome of their actions more easily.

Note that the metaphor can break down, however. What is the real-world equivalent of formatting a disk? Is there a direct analogy for the concept of 'undo'? Think of some more examples yourself.

RECOMMENDED READING

D. A. Norman, *The Psychology of Everyday Things*, Basic Books, 1988. (Republished as *The Design of Everyday Things* by Penguin, 1991.)
A classic text, which discusses psychological issues in designing everyday objects and addresses why such objects are often so difficult to use. Discusses the execution–evaluation cycle. Very readable and entertaining. See also his more recent books *Turn Signals are the Facial Expressions of Automobiles* [267], *Things That Make Us Smart* [268] and *The Invisible Computer* [269].

R. W. Bailey, *Human Performance Engineering: A Guide for System Designers*, Prentice Hall, 1982.
Detailed coverage of human factors and ergonomics issues, with plenty of examples.

G. Salvendy, *Handbook of Human Factors and Ergonomics*, John Wiley, 1997.
Comprehensive collection of contributions from experts in human factors and ergonomics.

M. Helander, editor, *Handbook of Human–Computer Interaction. Part II: User Interface Design*, North-Holland, 1988.
Comprehensive coverage of interface styles.

J. Raskin, *The Humane Interface: New Directions for Designing Interactive Systems*, Addison Wesley, 2000.
Jef Raskin was one of the central designers of the original Mac user interface. This book gives a personal, deep and practical examination of many issues of interaction and its application in user interface design.

M. Blythe, A. Monk, K. Overbeeke and P. Wright, editors, *Funology: From Usability to Enjoyment*, Kluwer, 2003.
This is an edited book with chapters covering many areas of user experience. It includes an extensive review of theory from many disciplines from psychology to literary theory and chapters giving design frameworks based on these. The theoretical and design base is grounded by many examples and case studies including a detailed analysis of virtual crackers.

4

Paradigms

Overview

- Examples of effective strategies for building interactive systems provide paradigms for designing usable interactive systems.
- The evolution of these usability paradigms also provides a good perspective on the history of interactive computing.
- These paradigms range from the introduction of time-sharing computers, through the WIMP and web, to ubiquitous and context-aware computing.

4.1 INTRODUCTION

As we noted in Chapter 3, the primary objective of an interactive system is to allow the user to achieve particular goals in some application domain, that is, the interactive system must be usable. The designer of an interactive system, then, is posed with two open questions:

1. How can an interactive system be developed to ensure its usability?
2. How can the usability of an interactive system be demonstrated or measured?

One approach to answering these questions is by means of example, in which successful interactive systems are commonly believed to enhance usability and, therefore, serve as *paradigms* for the development of future products.

We believe that we now build interactive systems that are more usable than those built in the past. We also believe that there is considerable room for improvement in designing more usable systems in the future. As discussed in Chapter 2, the great advances in computer technology have increased the power of machines and enhanced the bandwidth of communication between humans and computers. The impact of technology alone, however, is not sufficient to enhance its usability. As our machines have become more powerful, the key to increased usability has come from the creative and considered application of the technology to accommodate and augment the power of the human. Paradigms for interaction have for the most part been dependent upon technological advances and their creative application to enhance interaction.

In this chapter, we investigate some of the principal historical advances in interactive designs. What is important to notice here is that the techniques and designs mentioned are recognized as major improvements in interaction, though it is sometimes hard to find a consensus for the reason behind the success. It is even harder to predict ahead what the new paradigms will be. Often new paradigms have arisen through exploratory designs that have then been seen, after the fact, to have created a new base point for future design.

We will discuss 15 different paradigms in this chapter. They do not provide mutually exclusive categories, as particular systems will often incorporate ideas from more than one of the following paradigms. In a way, this chapter serves as a history of interactive system development, though our emphasis is not so much on historical accuracy as on interactive innovation. We are concerned with the advances in interaction provided by each paradigm.

4.2 PARADIGMS FOR INTERACTION

4.2.1 Time sharing

In the 1940s and 1950s, the significant advances in computing consisted of new hardware technologies. Mechanical relays were replaced by vacuum electron tubes. Tubes were replaced by transistors, and transistors by integrated chips, all of which meant that the amount of sheer computing power was increasing by orders of magnitude. By the 1960s it was becoming apparent that the explosion of growth in computing power would be wasted if there was not an equivalent explosion of ideas about how to channel that power. One of the leading advocates of research into human-centered applications of computer technology was J. C. R. Licklider, who became the director of the Information Processing Techniques Office of the US Department of Defense's Advanced Research Projects Agency (ARPA). It was Licklider's goal to finance various research centers across the United States in order to encourage new ideas about how best to apply the burgeoning computing technology.

One of the major contributions to come out of this new emphasis in research was the concept of *time sharing*, in which a single computer could support multiple users. Previously, the human (or more accurately, the programmer) was restricted to batch sessions, in which complete jobs were submitted on punched cards or paper tape to an operator who would then run them individually on the computer. Time-sharing systems of the 1960s made programming a truly interactive venture and brought about a subculture of programmers known as 'hackers' – single-minded masters of detail who took pleasure in understanding complexity. Though the purpose of the first interactive time-sharing systems was simply to augment the programming capabilities of the early hackers, it marked a significant stage in computer applications for human use. Rather than rely on a model of interaction as a pre-planned activity that resulted in a complete set of instructions being laid out for the computer to follow, truly interactive exchange between programmer and computer was possible. The computer could now project itself as a dedicated partner with each individual user and the increased throughput of information between user and computer allowed the human to become a more reactive and spontaneous collaborator. Indeed, with the advent of time sharing, real human–computer interaction was now possible.

4.2.2 Video display units

As early as the mid-1950s researchers were experimenting with the possibility of presenting and manipulating information from a computer in the form of images on a video display unit (VDU). These display screens could provide a more suitable medium than a paper printout for presenting vast quantities of strategic information for rapid assimilation. The earliest applications of display screen images were developed in military applications, most notably the Semi-Automatic Ground Environment (SAGE) project of the US Air Force. It was not until 1962, however, when a young graduate student at the Massachusetts Institute of Technology (MIT), Ivan Sutherland, astonished the established computer science community with his *Sketchpad* program, that the capabilities of visual images were realized. As described in Howard Rheingold's history of computing book *Tools for Thought* [305]:

> Sketchpad allowed a computer operator to use the computer to create, very rapidly, sophisticated visual models on a display screen that resembled a television set. The visual patterns could be stored in the computer's memory like any other data, and could be manipulated by the computer's processor. . . . But Sketchpad was much more

> than a tool for creating visual displays. It was a kind of simulation language that enabled computers to translate abstractions into perceptually concrete forms. And it was a model for totally new ways of operating computers; by changing something on the display screen, it was possible, via Sketchpad, to change something in the computer's memory.

Sketchpad demonstrated two important ideas. First, computers could be used for more than just data processing. They could extend the user's ability to abstract away from some levels of detail, visualizing and manipulating different representations of the same information. Those abstractions did not have to be limited to representations in terms of bit sequences deep within the recesses of computer memory. Rather, the abstractions could be made truly visual. To enhance human interaction, the information within the computer was made more amenable to human consumption. The computer was made to speak a more human language, instead of the human being forced to speak more like a computer. Secondly, Sutherland's efforts demonstrated how important the contribution of one creative mind (coupled with a dogged determination to see the idea through) could be to the entire history of computing.

4.2.3 Programming toolkits

Douglas Engelbart's ambition since the early 1950s was to use computer technology as a means of complementing human problem-solving activity. Engelbart's idea as a graduate student at the University of California at Berkeley was to use the computer to teach humans. This dream of naïve human users actually learning from a computer was a stark contrast to the prevailing attitude of his contemporaries that computers were a purposely complex technology that only the intellectually privileged were capable of manipulating. Engelbart's dedicated research team at the Stanford Research Institute in the 1960s worked towards achieving the manifesto set forth in an article published in 1963 [124]:

> By 'augmenting man's intellect' we mean increasing the capability of a man to approach a complex problem situation, gain comprehension to suit his particular needs, and to derive solutions to problems. . . . We refer to a way of life in an integrated domain where hunches, cut-and-try, intangibles, and the human 'feel for the situation' usefully coexist with powerful concepts, streamlined terminology and notation, sophisticated methods, and high-powered electronic aids.

Many of the ideas that Engelbart's team developed at the Augmentation Research Center – such as word processing and the mouse – only attained mass commercial success decades after their invention. A live demonstration of his oNLine System (NLS, also later known as NLS/Augment) was given in the autumn of 1968 at the Fall Joint Computer Conference in San Francisco before a captivated audience of computer sceptics. We are not so concerned here with the interaction techniques that were present in NLS, as many of those will be discussed later. What is important here is the method that Engelbart's team adopted in creating their very innovative and powerful interactive systems with the relatively impoverished technology of the 1960s.

Engelbart wrote of how humans attack complex intellectual problems like a carpenter who produces beautifully complicated pieces of woodwork with a good set of tools. The secret to producing computing equipment that aided human problem-solving ability was in providing the right *toolkit*. Taking this message to heart, his team of programmers concentrated on developing the set of programming tools they would require in order to build more complex interactive systems. The idea of building components of a computer system that will allow you to rebuild a more complex system is called bootstrapping and has been used to a great extent in all of computing. The power of programming toolkits is that small, well-understood components can be composed in fixed ways in order to create larger tools. Once these larger tools become understood, they can continue to be composed with other tools, and the process continues.

4.2.4 Personal computing

Programming toolkits provide a means for those with substantial computing skills to increase their productivity greatly. But Engelbart's vision was not exclusive to the computer literate. The decade of the 1970s saw the emergence of computing power aimed at the masses, computer literate or not. One of the first demonstrations that the powerful tools of the hacker could be made accessible to the computer novice was a graphics programming language for children called LOGO. The inventor, Seymour Papert, wanted to develop a language that was easy for children to use. He and his colleagues from MIT and elsewhere designed a computer-controlled mechanical turtle that dragged a pen along a surface to trace its path. A child could quite easily pretend they were 'inside' the turtle and direct it to trace out simple geometric shapes, such as a square or a circle. By typing in English phrases, such as `Go forward` or `Turn left`, the child/programmer could teach the turtle to draw more and more complicated figures. By adapting the graphical programming language to a model which children could understand and use, Papert demonstrated a valuable maxim for interactive system development – no matter how powerful a system may be, it will always be more powerful if it is easier to use.

Alan Kay was profoundly influenced by the work of both Engelbart and Papert. He realized that the power of a system such as NLS was only going to be successful if it was as accessible to novice users as was LOGO. In the early 1970s his view of the future of computing was embodied in small, powerful machines which were dedicated to single users, that is *personal computers*. Together with the founding team of researchers at the Xerox Palo Alto Research Center (PARC), Kay worked on incorporating a powerful and simple visually based programming environment, Smalltalk, for the personal computing hardware that was just becoming feasible. As technology progresses, it is now becoming more difficult to distinguish between what constitutes a personal computer, or workstation, and what constitutes a mainframe. Kay's vision in the mid-1970s of the ultimate handheld personal computer – he called it the Dynabook – outstrips even the technology we have available today [197].

4.2.5 Window systems and the WIMP interface

With the advent and immense commercial success of personal computing, the emphasis for increasing the usability of computing technology focussed on addressing the single user who engaged in a dialog with the computer in order to complete some work. Humans are able to think about more than one thing at a time, and in accomplishing some piece of work, they frequently interrupt their current train of thought to pursue some other related piece of work. A personal computer system which forces the user to progress in order through all of the tasks needed to achieve some objective, from beginning to end without any diversions, does not correspond to that standard working pattern. If the personal computer is to be an effective dialog partner, it must be as flexible in its ability to 'change the topic' as the human is.

But the ability to address the needs of a different user task is not the only requirement. Computer systems for the most part react to stimuli provided by the user, so they are quite amenable to a wandering dialog initiated by the user. As the user engages in more than one plan of activity over a stretch of time, it becomes difficult for him to maintain the status of the overlapping threads of activity. It is therefore necessary for the computer dialog partner to present the context of each thread of dialog so that the user can distinguish them.

One presentation mechanism for achieving this dialog partitioning is to separate physically the presentation of the different logical threads of user–computer conversation on the display device. The *window* is the common mechanism associated with these physically and logically separate display spaces. We discussed windowing systems in detail in Chapter 3.

Interaction based on windows, icons, menus and pointers – the WIMP interface – is now commonplace. These interaction devices first appeared in the commercial marketplace in April 1981, when Xerox Corporation introduced the 8010 Star Information System. But many of the interaction techniques underlying a windowing system were used in Engelbart's group in NLS and at Xerox PARC in the experimental precursor to Star, the Alto.

4.2.6 The metaphor

In developing the LOGO language to teach children, Papert used the metaphor of a turtle dragging its tail in the dirt. Children could quickly identify with the real-world phenomenon and that instant familiarity gave them an understanding of how they could create pictures. Metaphors are used quite successfully to teach new concepts in terms of ones which are already understood, as we saw when looking at analogy in Chapter 1. It is no surprise that this general teaching mechanism has been successful in introducing computer novices to relatively foreign interaction techniques. We have already seen how metaphors are used to describe the functionality of many interaction widgets, such as windows, menus, buttons and palettes. Tremendous commercial successes in computing have arisen directly from a judicious choice of metaphor. The Xerox Alto and Star were the first workstations based on the metaphor of the office desktop. The majority of the management tasks on a standard workstation have to do with file manipulation. Linking the set of tasks associated with file manipulation to the filing tasks in a typical office environment makes the actual computerized tasks easier to understand at first. The success of the desktop metaphor is unquestionable. Another good example in the personal computing domain is the widespread use of the spreadsheet metaphor for accounting and financial modeling.

Very few will debate the value of a good metaphor for increasing the initial familiarity between user and computer application. The danger of a metaphor is usually realized after the initial honeymoon period. When word processors were first introduced, they relied heavily on the typewriter metaphor. The keyboard of a computer closely resembles that of a standard typewriter, so it seems like a good metaphor from which to start. However, the behavior of a word processor is different from any typewriter. For example, the space key on a typewriter is passive, producing nothing on the piece of paper and just moving the guide further along the current line. For a typewriter, a space is not a character. However, for a word processor, the blank space *is* a character which must be inserted within a text just as any other character is inserted. So an experienced typist is not going to be able to predict correctly the behavior of pressing the spacebar on the keyboard by appealing to his experience with a typewriter. Whereas the typewriter metaphor is beneficial for providing a preliminary understanding of a word processor, the analogy is inadequate for promoting a full understanding of how the word processor works. In fact, the metaphor gets in the way of the user understanding the computer.

A similar problem arises with most metaphors. Although the desktop metaphor is initially appealing, it falls short in the computing world because there are no office equivalents for ejecting a floppy disk or printing a document. When designers try too hard to make the metaphor stick, the resulting system can be more confusing. Who thinks it is intuitive to drag the icon of a floppy disk to the wastebasket in order to eject it from the system? Ordinarily, the wastebasket is used to dispose of things that we never want to use again, which is why it works for deleting files. We must accept that some of the tasks we perform with a computer do not have real-world equivalents, or if they do, we cannot expect a single metaphor to account for all of them.

Another problem with a metaphor is the cultural bias that it portrays. With the growing internationalization of software, it should not be assumed that a metaphor will apply across national boundaries. A meaningless metaphor will only add another layer of complexity between the user and the system.

A more extreme example of metaphor occurs with *virtual reality* systems. In a VR system, the metaphor is not simply captured on a display screen. Rather, the user is also portrayed within the metaphor, literally creating an alternative, or virtual, reality. Any actions that the user performs are supposed to become more natural and so more movements of the user are interpreted, instead of just keypresses, button clicks and movements of an external pointing device. A VR system also needs to know the location and orientation of the user. Consequently, the user is often ‘rigged’ with special tracking devices so that the system can locate them and interpret their motion correctly.

4.2.7 Direct manipulation

In the early 1980s as the price of fast and high-quality graphics hardware was steadily decreasing, designers were beginning to see that their products were gaining popularity as their visual content increased. As long as the user–system dialog remained largely unidirectional – from user command to system command line prompt – computing was going to stay within the minority population of the hackers who revelled in the challenge of complexity. In a standard command line interface, the only way to get any feedback on the results of previous interaction is to know that you have to ask for it and to know how to ask for it. Rapid visual and audio *feedback* on a high-resolution display screen or through a high-quality sound system makes it possible to provide evaluative information for every executed user action.

Rapid feedback is just one feature of the interaction technique known as *direct manipulation*. Ben Shneiderman [320, 321] is attributed with coining this phrase in 1982 to describe the appeal of graphics-based interactive systems such as Sketchpad and the Xerox Alto and Star. He highlights the following features of a direct manipulation interface:

- visibility of the objects of interest
- incremental action at the interface with rapid feedback on all actions
- reversibility of all actions, so that users are encouraged to explore without severe penalties
- syntactic correctness of all actions, so that every user action is a legal operation
- replacement of complex command languages with actions to manipulate directly the visible objects (and, hence, the name direct manipulation).

The first real commercial success which demonstrated the inherent usability of direct manipulation interfaces for the general public was the Macintosh personal computer, introduced by Apple Computer, Inc. in 1984 after the relatively unsuccessful marketing attempt in the business community of the similar but more pricey Lisa computer. We discussed earlier how the desktop metaphor makes the computer domain of file management, usually described in terms of files and directories, easier to grasp by likening it to filing in the typical office environment, usually described in terms of documents and folders. The direct manipulation interface for the desktop metaphor requires that the documents and folders are made visible to the user as icons which represent the underlying files and directories. An operation such as moving a file from one directory to another is mirrored as an action on the visible document which is ‘picked up and dragged’ along the desktop from one folder to the next. In a command line interface to a filing system, it is normal that typographical errors in constructing the command line for a move operation would result in a syntactically incorrect command (for example, mistyping the file’s name results in an error if you are fortunate enough not to spell accidentally the name of another file in the process). It is impossible to formulate a syntactically incorrect move operation with the pick-up-and-drag style of command. It is still possible for errors to occur at a deeper level, as the user might move a document to the wrong place, but it is relatively easy to detect and recover from those errors. While the document is dragged, continual visual feedback is provided, creating the illusion that the user is actually working in the world of the desktop and not just using the metaphor to help him understand.

Ed Hutchins, Jim Hollan and Donald Norman [187] provide a more psychological justification in terms of the *model-world metaphor* for the directness that the above example suggests. In Norman and Draper's collection of papers on user-centered design [270] they write:

> In a system built on the model-world metaphor, the interface is itself a world where the user can act, and which changes state in response to user actions. The world of interest is explicitly represented and there is no intermediary between user and world. Appropriate use of the model-world metaphor can create the sensation in the user of acting upon the objects of the task domain themselves. We call this aspect of directness *direct engagement*.

In the model-world metaphor, the role of the interface is not so much one of mediating between the user and the underlying system. From the user's perspective, the interface *is* the system.

A consequence of the direct manipulation paradigm is that there is no longer a clear distinction between input and output. In the interaction framework in Chapter 3 we talked about a user articulating input expressions in some input language and observing the system-generated output expressions in some output language. In a direct manipulation system, the output expressions are used to formulate subsequent input expressions. The document icon is an output expression in the desktop metaphor, but that icon is used by the user to articulate the move operation. This aggregation of input and output is reflected in the programming toolkits, as widgets are not considered as input or output objects exclusively. Rather, widgets embody both input and output languages, so we consider them as *interaction objects*.

Somewhat related to the visualization provided by direct manipulation is the *WYSIWYG* paradigm, which stands for 'what you see is what you get'. What you see on a display screen, for example when you are using a word processor, is not the actual document that you will be producing in the end. Rather, it is a representation or rendering of what that final document will look like. The implication with a WYSIWYG interface is that the difference between the representation and the final product is minimal, and the user is easily able to visualize the final product from the computer's representation. So, in the word-processing example, you would be able to see what the overall layout of your document would be from its image on screen, minimizing any guesswork on your part to format the final printed copy.

With WYSIWYG interfaces, it is the simplicity and immediacy of the mapping between representation and final product that matters. In terms of the interaction framework, the observation of an output expression is made simple so that assessment of goal achievement is straightforward. But WYSIWYG is not a panacea for usability. What you see is all you get! In the case of a word processor, it is difficult to achieve more sophisticated page design if you must always see the results of the layout on screen. For example, suppose you want to include a picture in a document you are writing. You design the picture and then place it in the current draft of your document, positioning it at the top of the page on which it is first referenced. As you make changes to the paper, the position of the picture will change. If you still want it to appear at the top of a page, you will no doubt have to make adjustments to the document. It would be easier if you only had to include the picture once, with a directive that it should be positioned at the top of the printed page, whether or not it appears that way on screen. You might sacrifice the WYSIWYG principle in order to make it easier to incorporate such floatable objects in your documents.

Worked exercise *Discuss the ways in which a full-page word processor is or is not a direct manipulation interface for editing a document using Shneiderman's criteria. What features of a modern word processor break the metaphor of composition with pen (or typewriter) and paper?*

Answer We will answer the first point by evaluating the word processor relative to the criteria for direct manipulation given by Shneiderman.

Visibility of the objects of interest
The most important objects of interest in a word processor are the words themselves. Indeed, the visibility of the text on a continual basis was one of the major usability advances in moving from line-oriented to display-oriented editors. Depending on the user's application, there may be other objects of interest in word processing that may or may not be visible. For example, are the margins for the text on screen similar to the ones which would eventually be printed? Is the spacing within a line and the line breaks similar? Are the different fonts and formatting characteristics of the text visible (without altering the spacing)? Expressed in this way, we can see the visibility criterion for direct manipulation as very similar to the criteria for a WYSIWYG interface.

Incremental action at the interface with rapid feedback on all actions
We expect from a word processor that characters appear in the text as we type them in at the keyboard, with little delay. If we are inserting text on a page, we might also expect that the format of the page adjust immediately to accommodate the new changes. Various word processors do this reformatting immediately, whereas with others changes in page breaks may take some time to be reflected. One of the other important actions which requires incremental and rapid feedback is movement of the window using the scroll button. If there is a significant delay between the input command to move the window down and the actual movement of the window on screen, it is quite possible that the user will 'overshoot' the target when using the scrollbar button.

Reversibility of all actions, so that users are encouraged to explore without severe penalties
Single-step undo commands in most word processors allow the user to recover from the last action performed. One problem with this is that the user must recognize the error before doing any other action. More sophisticated undo facilities allow the user to retrace back more than one command at a time. The kind of exploration this reversibility provides in a word processor is best evidenced with the ease of experimentation that is now available for formatting changes in a document (font types and sizes and margin changes). One problem with the ease of exploration is that emphasis may move to the look of a document rather than what the text actually says (style over content).

Syntactic correctness of all actions, so that every user action is a legal operation
WYSIWYG word processors usually provide menus and buttons which the user uses to articulate many commands. These interaction mechanisms serve to constrain the input language to allow only legal input from the user. Document markup systems, such as HTML and LaTeX, force the user to insert textual commands (which may be erroneously entered by the user) to achieve desired formatting effects.

Replacement of complex command languages with actions to manipulate directly the visible objects
The case for word processors is similar to that described above for syntactic correctness. In addition, operations on portions of text are achieved many times by allowing the user to highlight the text directly with a mouse (or arrow keys). Subsequent action on that text, such as moving it or copying it to somewhere else, can then be achieved more directly by allowing the user to 'drag' the selected text via the mouse to its new location.

To answer the second question concerning the drawback of the pen (or typewriter) metaphor for word processing, we refer to the discussion on metaphors in Section 4.2.6. The example there compares the functionality of the space key in typewriting versus word processing. For a typewriter, the space key is passive; it merely moves the insertion point one space to the right. In a word processor, the space key is active, as it inserts a character (the space character) into the document. The functionality of the typewriter space key is produced by the movement keys for the word processor (typically an arrow key pointing right to move forward within one line). In fact, much of the functionality that we have come to expect of a word processor is radically different from that expected of a typewriter, so much so that the typewriter as a metaphor for word processing is not all that instructive. In practice, modern typewriters have begun to borrow from word processors when defining their functionality!

4.2.8 Language versus action

Whereas it is true that direct manipulation interfaces make some tasks easier to perform correctly, it is equally true that some tasks are more difficult, if not impossible. Contrary to popular wisdom, it is not generally true that actions speak louder than words. The image we projected for direct manipulation was of the interface as a replacement for the underlying system as the world of interest to the user. Actions performed at the interface replace any need to understand their meaning at any deeper, system level. Another image is of the interface as the interlocutor or mediator between the user and the system. The user gives the interface instructions and it is then the responsibility of the interface to see that those instructions are carried out. The user–system communication is by means of indirect language instead of direct actions.

We can attach two meaningful interpretations to this language paradigm. The first requires that the user understands how the underlying system functions and the interface as interlocutor need not perform much translation. In fact, this interpretation of the language paradigm is similar to the kind of interaction which existed before direct manipulation interfaces were around. In a way, we have come full circle!

The second interpretation does not require the user to understand the underlying system's structure. The interface serves a more active role, as it must interpret between the intended operation as requested by the user and the possible system operations that must be invoked to satisfy that intent. Because it is more active, some people refer to the interface as an *agent* in these circumstances. We can see this kind of language paradigm at work in an information retrieval system. You may know what kind of information is in some internal system database, such as the UK highway code, but you would not know how that information is organized. If you had a question about speed limits on various roads, how would you ask? The answer in this case is that you would ask the question in whatever way it comes to mind, typing in a question such as, 'What are the speed limits on different roads?' You then leave it up to the interface agent to reinterpret your request as a legal query to the highway code database.

Whatever interpretation we attach to the language paradigm, it is clear that it has advantages and disadvantages when compared with the action paradigm implied by direct manipulation interfaces. In the action paradigm, it is often much easier to perform simple tasks without risk of certain classes of error. For example, recognizing and pointing to an object reduces the difficulty of identification and the possibility of misidentification. On the other hand, more complicated tasks are often rather tedious to perform in the action paradigm, as they require repeated execution of the same procedure with only minor modification. In the language paradigm, there is the possibility of describing a generic procedure once (for example, a looping construct which will perform a routine manipulation on all files in a directory) and then leaving it to be executed without further user intervention.

The action and language paradigms need not be completely separate. In the above example, we distinguished between the two paradigms by saying that we can describe generic and repeatable procedures in the language paradigm and not in the action paradigm. An interesting combination of the two occurs in *programming by example* when a user can perform some routine tasks in the action paradigm and the system records this as a generic procedure. In a sense, the system is interpreting the user's actions as a language script which it can then follow.

4.2.9 Hypertext

In 1945, Vannevar Bush, then the highest-ranking scientific administrator in the US war effort, published an article entitled 'As We May Think' in *The Atlantic Monthly*. Bush was in charge of over 6000 scientists who had greatly pushed back the frontiers of scientific knowledge during the Second World War. He recognized that a major drawback of these prolific research efforts was that it was becoming increasingly difficult to keep in touch with the growing body of scientific knowledge in the literature. In his opinion, the greatest advantages of this scientific revolution were to be gained by those individuals who were able to keep abreast of an ever-increasing flow of information. To that end, he described an innovative and futuristic information storage and retrieval apparatus – the *memex* – which was constructed with technology wholly existing in 1945 and aimed at increasing the human capacity to store and retrieve connected pieces of knowledge by mimicking our ability to create random associative links.

The memex was essentially a desk with the ability to produce and store a massive quantity of photographic copies of documented information. In addition to its huge storage capacity, the memex could keep track of links between parts of different documents. In this way, the stored information would resemble a vast interconnected mesh of data, similar to how many perceive information is stored in the human brain. In the context of scientific literature, where it is often very difficult to keep track of the origins and interrelations of the ever-growing body of research, a device which explicitly stored that information would be an invaluable asset.

We have already discussed some of the contributions of 'disciples' of Bush's vision – Douglas Engelbart and Alan Kay. One other follower was equally influenced by the ideas behind the memex, though his dreams have not yet materialized to the extent of Engelbart's and Kay's. Ted Nelson was another graduate student/dropout whose research agenda was forever transformed by the advent of the computer. An unsuccessful attempt to create a machine language equivalent of the memex on early 1960s computer hardware led Nelson on a lifelong quest to produce *Xanadu*, a potentially revolutionary worldwide publishing and information retrieval system based on the idea of interconnected, non-linear text and other media forms. A traditional paper is read from beginning to end, in a linear fashion. But within that text, there are often ideas or footnotes that urge the reader to digress into a richer topic. The linear format for information does not provide much support for this random and associated browsing task. What Bush's memex suggested was to preserve the non-linear browsing structure in the actual documentation. Nelson coined the phrase *hypertext* in the mid-1960s to reflect this non-linear text structure.

It was nearly two decades after Nelson coined the term that the first hypertext systems came into commercial use. In order to reflect the use of such non-linear and associative linking schemes for more than just the storage and retrieval of textual information, the term *hypermedia* (or *multimedia*) is used for non-linear storage of all forms of electronic media. We will discuss these systems in Part 4 of this book (see Chapter 21). Most of the riches won with the success of hypertext and hypermedia were not gained by Nelson, though his project Xanadu survives to this day.

4.2.10 Multi-modality

The majority of interactive systems still use the traditional keyboard and a pointing device, such as a mouse, for input and are restricted to a color display screen with some sound capabilities for output. Each of these input and output devices can be considered as communication channels for the system and they correspond to certain human communication channels, as we saw in Chapter 1. A *multi-modal* interactive system is a system that relies on the use of multiple human communication channels. Each different channel for the user is referred to as a modality of interaction. In this sense, all interactive systems can be considered multi-modal, for humans have always used their visual and haptic (touch) channels in manipulating a computer. In fact, we often use our audio channel to hear whether the computer is actually running properly.

However, genuine multi-modal systems rely to a greater extent on simultaneous use of multiple communication channels for both input and output. Humans quite naturally process information by simultaneous use of different channels. We point to someone and refer to them as 'you', and it is only by interpreting the simultaneous use of voice and touch that our directions are easily articulated and understood. Designers have wanted to mimic this flexibility in both articulation and observation by extending the input and output expressions an interactive system will support. So, for example, we can modify a gesture made with a pointing device by speaking, indicating what operation is to be performed on the selected object.

Multi-modal, multimedia and virtual reality systems form a large core of current research in interactive system design. These are discussed in more detail in Chapters 10, 20 and 21.

4.2.11 Computer-supported cooperative work

Another development in computing in the 1960s was the establishment of the first computer networks which allowed communication between separate machines. Personal computing was all about providing individuals with enough computing power so that they were liberated from dumb terminals which operated on a time-sharing system. It is interesting to note that as computer networks became widespread, individuals retained their powerful workstations but now wanted to reconnect themselves to the rest of the workstations in their immediate working environment, and even throughout the world! One result of this reconnection was the emergence of collaboration between individuals via the computer – called computer-supported cooperative work, or CSCW.

The main distinction between CSCW systems and interactive systems designed for a single user is that designers can no longer neglect the society within which any single user operates. CSCW systems are built to allow interaction between humans via the computer and so the needs of the many must be represented in the one product. A fine example of a CSCW system is electronic mail – *email* – yet another metaphor by which individuals at physically separate locations can communicate via electronic messages which work in a similar way to conventional postal systems. One user can compose a message and 'post' it to another user (specified by his electronic mail address). When the message arrives at the remote user's site, he is informed that a new message has arrived in his 'mailbox'. He can then read the message and respond as desired. Although email is modeled after conventional postal systems, its major advantage is that it is often much faster than the traditional system (jokingly referred to by email devotees as 'snail mail'). Communication turnarounds between sites across the world are in the order of minutes, as opposed to weeks.

Electronic mail is an instance of an asynchronous CSCW system because the participants in the electronic exchange do not have to be working at the same time in order for the mail to be delivered. The reason we use email is precisely because of its asynchronous characteristics. All we need to know is that the recipient will eventually receive the message. In contrast, it might be desirable for synchronous communication, which would require the simultaneous participation of sender and recipient, as in a phone conversation.

CSCW is a major emerging topic in current HCI research, and so we devote much more attention to it later in this book. CSCW systems built to support users working in groups are referred to as *groupware*. Chapter 19 discusses groupware systems in depth. In Chapter 14 the more general issues and theories arising from CSCW are discussed.

4.2.12 The world wide web

Probably the most significant recent development in interactive computing is the world wide web, often referred to as just the web, or WWW. The web is built on top of the internet, and offers an easy to use, predominantly graphical interface to information, hiding the underlying complexities of transmission protocols, addresses and remote access to data.

The internet (see Section 2.9) is simply a collection of computers, each linked by any sort of data connection, whether it be slow telephone line and modem or high-bandwidth optical connection. The computers of the internet all communicate using common data transmission protocols (*TCP/IP*) and addressing systems (*IP* addresses and *domain names*). This makes it possible for anyone to read anything from anywhere, in theory, if it conforms to the protocol. The web builds on this with its own layer of network protocol (http), a standard markup notation (such as HTML) for laying out pages of information and a global naming scheme (uniform resource locators or URLs). Web pages can contain text, color images, movies, sound and, most important, hypertext links to other web pages. Hypermedia documents can therefore be 'published' by anyone who has access to a computer connected to the internet.

The world wide web project was conceived in 1989 by Tim Berners-Lee, working at CERN, the European Particle Physics Laboratory at Geneva, as a means to enable the widespread distribution of scientific data generated at CERN and to share information between physicists worldwide. In 1991 the first text-based web browser was released. This was followed in early 1993 by several graphical web browsers, most significantly Mosaic developed by Marc Andreesen at the National Center for Supercomputer Applications (NCSA) at Champaign, Illinois. This was the defining moment at which the meteoric growth of the web began, rapidly growing to dominate internet traffic and change the public view of computing. Of all the 'heroes' of interactive computing named in this chapter, it is only Berners-Lee who has achieved widespread public fame.

Whilst the internet has been around since 1969, it did not become a major paradigm for interaction until the advent and ease of availability of well-designed graphical interfaces (browsers) for the web. These browsers allow users to access multimedia information easily, using only a mouse to point and click. This shift towards the integration of computation and communication is transparent to users; all they realize is that they can get the current version of published information practically instantly. In addition, the language used to create these multimedia documents is relatively simple, opening the opportunity of publishing information to any literate, and connected, person. However, there are important limitations of the web as a hypertext medium and in Chapter 21 we discuss some of the special design issues for the web. Interestingly, the web did not provide any technological breakthroughs; all the required functionality previously existed, such as transmission protocols, distributed file systems, hypertext and so on. The impact has been due to the ease of use of both the browsers and HTML, and the fact that *critical mass* (see Chapter 13) was established, first in academic circles, and then rapidly expanded into the leisure and business domains. The burgeoning interest led to service providers, those providing connections to the internet, to make it cheap to connect, and a whole new subculture was born.

Currently, the web is one of the major reasons that new users are connecting to the internet (probably even buying computers in the first place), and is rapidly becoming a major activity for people both at work and for leisure. It is much more a social phenomenon than anything else, with users attracted to the idea that computers are now boxes that connect them with interesting people and exciting places to go, rather than soulless cases that deny social contact. Computing often used to be seen as an anti-social activity; the web has challenged this by offering a 'global village' with free access to information and a virtual social environment. Web culture has emphasized liberality and (at least in principle) equality regardless of gender, race and disability. In practice, the demographics of web users are only now coming close to equal proportions in terms of gender, and, although internet use is increasing globally, the vast majority of websites are still hosted in the United States. Indeed, the web is now big business; corporate images and e-commerce may soon dominate the individual and often zany aspects of the web.

4.2.13 Agent-based interfaces

In the human world agents are people who work on someone's behalf: estate agents buy and sell property for their customers, literary agents find publishers for authors, travel agents book hotels and air tickets for tourists and secret agents obtain information (secretly) for their governments. Software agents likewise act on behalf of users within the electronic world. Examples include email agents which filter your mail for you and web crawlers which search the world wide web for documents you might find interesting. Agents can perform repetitive tasks, watch and respond to events when the user is not present and even learn from the user's own actions.

Some agents simply do what they are told. For example, many email systems allow you to specify filters, simple *if then* rules, which determine the action to perform on certain kinds of mail message:

If Sender: is bank manager
Then Urgency: is high

A major problem with such agents is developing a suitable language between human and agent which allows the user to express intentions. This is especially important when the agent is going to act in the user's absence. In this case, the user may not receive feedback of any mistake until long after the effects have become irreversible; hence the instructions have to be correct, and believed to be correct.

Other agents use artificial intelligence techniques to learn based on the user's actions. An early example of this was Eager [83]. Eager watches users while they work on simple HyperCard applications. When it notices that the user is repeating similar actions a small icon appears (a smiling cat!), suggesting the next action. The user is free either to accept the suggestion or to ignore it. When the user is satisfied that Eager knows what it is doing, it can be instructed to perform all the remaining actions in a sequence.

Eager is also an example of an agent, which has a clear *embodiment*, that is, there is a representation of Eager (the cat icon) in the interface. In contrast, consider Microsoft Excel which incorporates some intelligence in its sum (Σ) function. If the current cell is directly below a column of numbers, or if there is a series of numbers to the left of the current cell, the sum range defaults to be the appropriate cells. It is also clever about columns of numbers with subtotals so that they are not included twice in the overall total. As around 80% of all spreadsheet formulae are simple sums this is a very useful feature. However, the intelligence in this is not embodied, it is diffuse, somewhere in 'the system'. Although embodiment is not essential to an agent-based system it is one of the key features which enable users to determine where autonomy and intelligence may lie, and also which parts are stable [107].

We have already discussed the relationship between language and action paradigms in human–computer interaction. To some extent agent-based systems include aspects of both. Old command-based systems acted as intermediaries: you asked them to do something, they did what you wanted (if you were lucky), and then reported the results back to you. In contrast, direct manipulation emphasizes the user's own actions, possibly augmented by tools, on the electronic world. Agents act on the user's behalf, possibly, but not necessarily, instructed in a linguistic fashion. But unlike the original intermediary paradigm, an agent is typically acting within a world the user could also act upon. The difference is rather like that between a traditional shopkeeper who brings items to you as opposed to a shop assistant in a supermarket who helps you as you browse amongst the aisles. The latter does not prevent you from selecting your own items from the shelves, but aids you when asked.

In fact, the proponents of direct manipulation and agent-based systems do not see the paradigms as being quite as complementary as we have described them above. Although amicable, the positions on each side are quite entrenched.

4.2.14 Ubiquitous computing

Where does computing happen, and more importantly, where do we as users go to interact with a computer? The past 50 years of interactive computing show that we still think of computers as being confined to a box on a desk or in an office or lab. The actual form of the physical interface has been transformed from a noisy teletype terminal to a large, graphical display with a WIMP or natural language interface, but in all cases the user knows where the computer is and must walk over to it to begin interacting with it.

In the late 1980s, a group of researchers at Xerox PARC, led by Mark Weiser, initiated a research program with the goal of moving human–computer interaction away from the desktop and out into our everyday lives. Weiser observed:

> The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.

These words have inspired a new generation of researchers in the area of *ubiquitous computing* [369, 370]. Another popular term for this emerging paradigm is *pervasive* computing, first coined by IBM. The intention is to create a computing infrastructure that permeates our physical environment so much that we do not notice the computer any longer. A good analogy for the vision of ubiquitous computing is the electric motor. When the electric motor was first introduced, it was large, loud and very noticeable. Today, the average household contains so many electric motors that we hardly ever notice them anymore. Their utility led to ubiquity and, hence, invisibility.

How long in the future will it be before we no longer notice the interactive computer? To some extent, this is already happening, since many everyday items, such as watches, microwaves or automobiles, contain many microprocessors that we don't directly notice. But, to a large extent, the vision of Weiser, in which the computer is hardly ever noticed, is a long way off.

To pursue the analogy with the electric motor a little further, one of the motor's characteristics is that it comes in many sizes. Each size is suited to a particular use. Weiser thought that it was also important to think of computing technology in different sizes. The original work at PARC looked at three different scales of computing: the yard, the foot and the inch. In the middle of the scale, a foot-sized computer is much like the personal computers we are familiar with today. Its size is suitable for every individual to have one, perhaps on their desk or perhaps in their bedroom or in their briefcase. A yard-sized computer, on the other hand, is so large that it would be suitable for wide open public spaces, and would be shared by a group of people. Perhaps there would be one of these in every home, or in a public hallway or auditorium. On the opposite side of the scale, an inch-sized computer would be a truly personal computing device that could fit in the palm of a hand. Everyone would have a number of these at their disposal, and they would be as prevalent and unremarkable as a pen or a pad of sticky notes.

There is an increasing number of examples of computing devices at these different scales. At the foot scale, laptop computers are, of course, everywhere, but more interesting examples of computing at this scale are commercially available *tablet computers* or research prototypes, such as an interactive storybook (see Figure 4.1). At the yard scale, there are various forms of high-resolution large screens and projected displays as we discussed in Chapter 2 (Section 2.4.3). These are still mainly used as output-only devices showing presentations or fixed messages, but there is

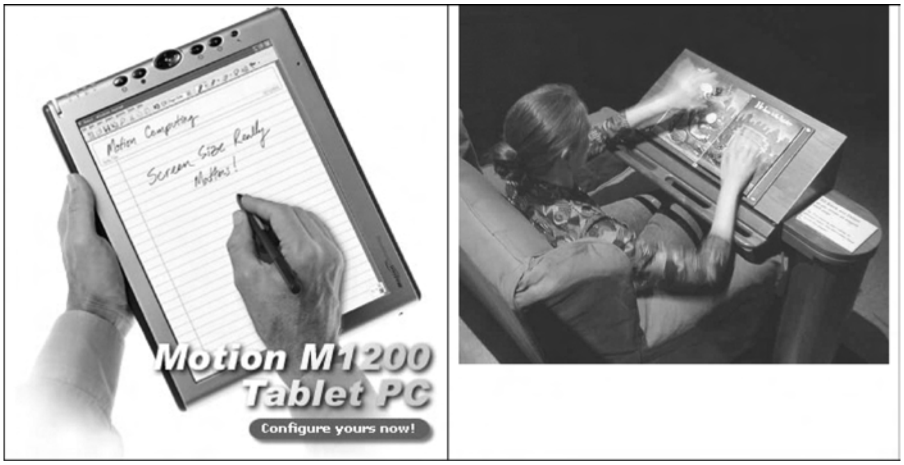


Figure 4.1 Examples of computing devices at the foot scale. On the left is a tablet computer – a Tablet PC from MotionComputing (Source: Motion Computing, Inc.). On the right is a research prototype, the Listen Reader, an interactive storybook developed at Palo Alto Research Center (picture courtesy Palo Alto Research Center)

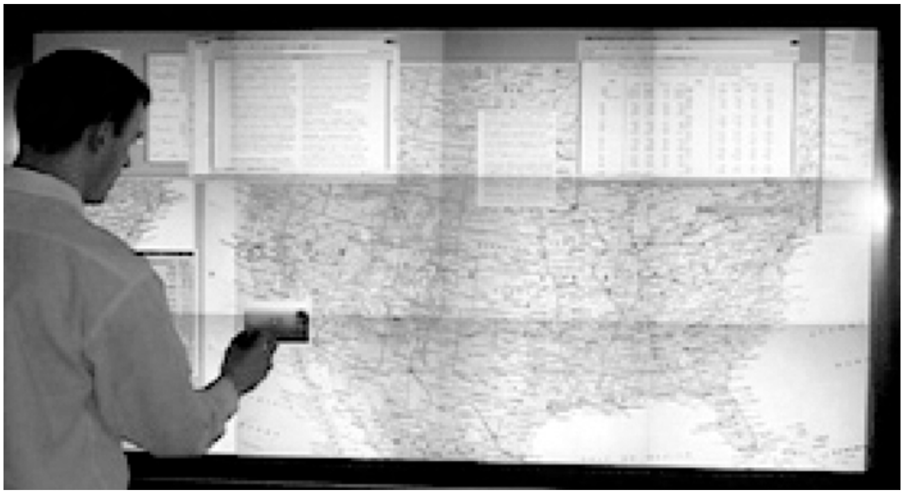


Figure 4.2 The Stanford Interactive Mural, an example of a yard-scale interactive display surface created by tiling multiple lower-resolution projectors. Picture courtesy François Guimbretière

increasing use of more interactive shared public displays, such as the Stanford Interactive Mural shown in Figure 4.2. At the inch scale, there are many examples, from powerful, pocket-sized *personal organizers* or *personal digital assistants* (PDAs) to even smaller cellular phones or pagers, and many pocket electronic devices such as electronic dictionaries and translators (see Figure 4.3). There are even badges whose position can be automatically tracked.

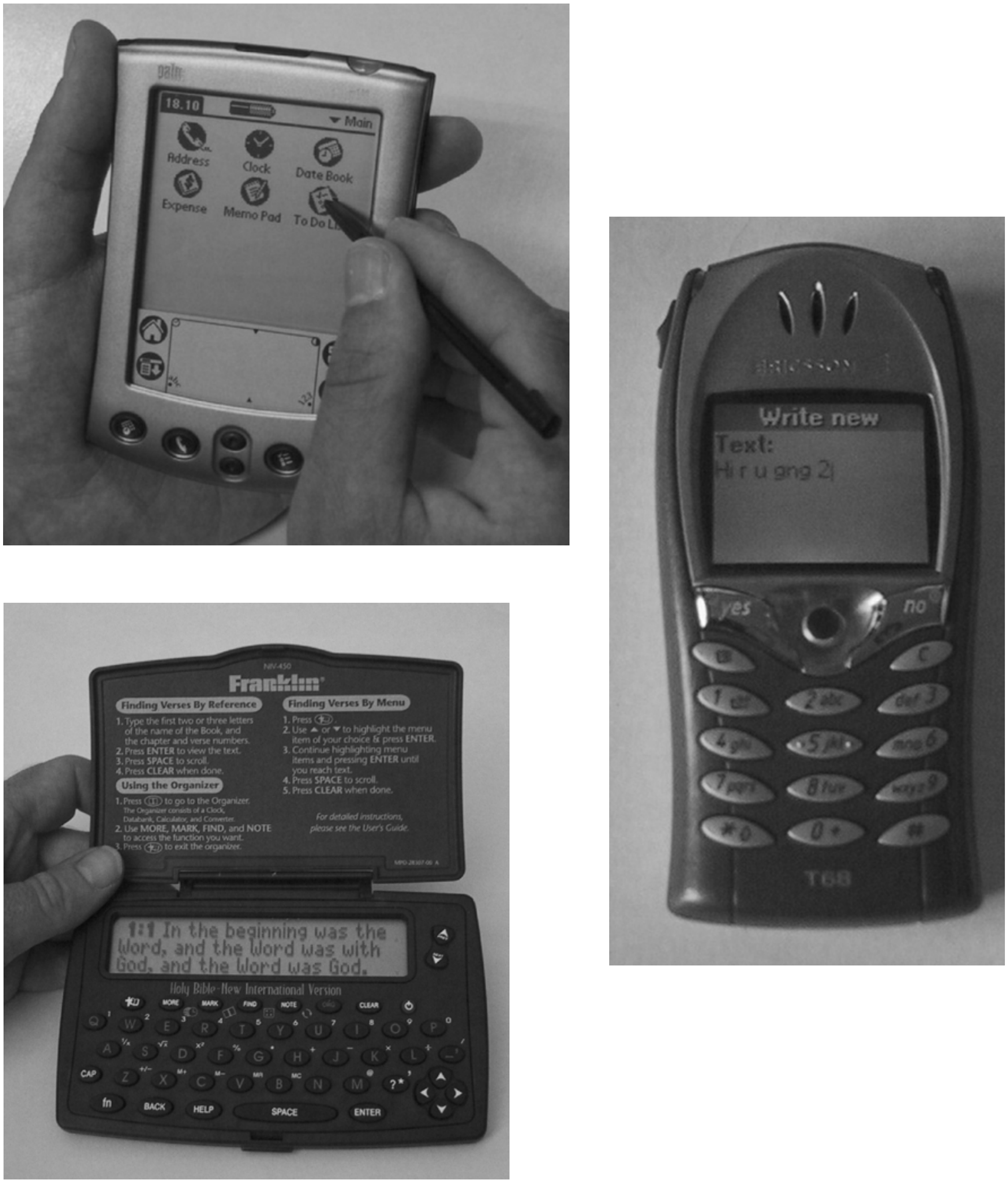

18.10
Main
Address
Clock
Date Book
Expense
Memo Pad
To Do List
ERICSSON
Write new
Text:
yes
no
T68
Franklin
Finding Verses By Reference
Finding Verses By Menu
Using the Organizer
1:1 In the beginning was the Word, and the Word was with God, and the Word was God.
Holy Bible-New International Version
SPACE
ENTER

Figure 4.3 Example inch-scale devices. From left to right, a PDA, a mobile phone and pocket-sized electronic bible. Source: Top left photograph by Alan Dix (Palm Pilot Series V), bottom left photograph by Alan Dix with permission from Franklin Electronic Publishers, photograph right by Alan Dix (Ericsson phone)

This influx of diverse computing devices represents the *third wave of computing*, in which the ratio of computers to human drastically changes. In the first wave of computing, one large mainframe computer served many people. In the second wave, the PC revolution, computing devices roughly equalled the number of people using them. In the third wave, the devices far outnumber the people. It is precisely because of the large ratio of devices to people that Weiser and others note the importance of minimizing the attention demands of any single device.

Many different technologies are converging to make the dream of ubiquitous computing possible. These technologies include wireless networking, voice recognition, camera and vision systems, pen-based computing and positioning systems, to name a few. What all of these technologies provide is the ability to move the computer user away from a desktop, allow interaction in a number of modes (voice, gesture, handwriting) in addition to a keyboard, and make information about the user (through vision, speech recognition or positioning information) available to a computational device that may be far removed from the actual user.

Ubiquitous computing is not simply about nifty gadgets, it is what can be done with those gadgets. As Weiser pointed out, it is the applications that make ubiquitous computing revolutionary. In Chapter 20, we discuss some examples of the applications that ubiquitous computing makes possible, including the way this is becoming part of everyday life in places as diverse as the home, the car and even our own bodies. The vision of ubiquitous computing – first expressed by Weiser and grounded in experimental work done at Xerox PARC – is now starting to become reality.

4.2.15 Sensor-based and context-aware interaction

The yard-scale, foot-scale and inch-scale computers are all still clearly embodied devices with which we interact, whether or not we consider them 'computers'. There are an increasing number of proposed and existing technologies that embed computation even deeper, but unobtrusively, into day-to-day life. Weiser's dream was computers that 'permeate our physical environment so much that we do not notice the computers anymore', and the term ubiquitous computing encompasses a wide range from mobile devices to more pervasive environments.

We can consider the extreme situation in which the user is totally unaware of interaction taking place. Information can be gathered from sensors in the environment (pressure mats, ultrasonic movement detectors, weight sensors, video cameras), in our information world (web pages visited, times online, books purchased online), and even from our own bodies (heart rate, skin temperature, brain signals). This information is then used by systems that make inferences about our past patterns and current *context* in order to modify the explicit interfaces we deal with (e.g., modify default menu options) or to do things in the background (e.g., adjust the air conditioning).

We already encounter examples of this: lights that turn on when we enter a room, suggestions made for additional purchases at online bookstores, automatic doors and washbasins. For elderly and disabled people, assistive technologies already embody quite radical aspects of this. However, the vision of many is a world in which the whole environment is empowered to sense and even understand the context of activities within it.

Previous interactive computation has focussed on the user explicitly telling the computer exactly what to do and the computer doing what it is told. In *context-aware computing* the interaction is more implicit. The computer, or more accurately the sensor-enhanced environment, is using heuristics and other semi-intelligent means to predict what would be useful for the user. The data used for this inference and the inference itself are both fuzzy, probabilistic and uncertain. Automatically sensing context is, and will likely always remain, an imperfect activity, so it is important that the actions resulting from these 'intelligent' predictions be made with caution. Context-aware applications should follow the principles of *appropriate intelligence*:

1. Be right as often as possible, and useful when acting on these correct predictions.
2. Do not cause inordinate problems in the event of an action resulting from a wrong prediction.

The failure of 'intelligent' systems in the past resulted from following the first principle, but not the second. These new applications, which impinge so closely on our everyday lives, demand that the second principle of appropriate intelligence is upheld. (There is more on using intelligence in interfaces on the book website at /e3/online/intelligence/)

Arguably this is a more radical paradigm shift than any other since the introduction of interactive computing itself. Whereas ubiquitous computing challenges the idea of *where* computers are and how apparent they are to us, context-aware computing challenges *what it means to interact* with a computer. It is as if we have come full circle from the early days of computing. Large mainframes were placed in isolation from the principle users (programmers) and interaction was usually done through an intermediary operator. Half a century later, the implicit nature of interaction implied by sensing creates a human–computer relationship that becomes so seamless there is no conscious interaction at all.

This shift is so radical that one could even say it does not belong in this chapter about paradigms for interaction! In fact, this shift is so dramatic that it is unclear whether the basic models of interaction that have proved universal across technologies, for example Norman's execution–evaluation cycle (Chapter 3, Section 3.2.2), are applicable at all. We will return to this issue in Chapter 18.

4.3 SUMMARY

In this chapter, we have discussed paradigms that promote the usability of interactive systems. We have seen that the history of computing is full of examples of creative insight into how the interaction between humans and computers can be enhanced. While we expect never to replace the input of creativity in interactive system design, we still want to maximize the benefit of one good idea by repeating its benefit in many other designs. The problem with these paradigms is that they are rarely well defined. It is not always clear how they support a user in accomplishing some tasks. As a result, it is entirely possible that repeated use of some paradigm will not result in the design of a more usable system. The derivation of principles and theoretical models for interaction has often arisen out of a need to explain why a paradigm is successful and when it might not be. Principles can provide the repeatability that paradigms in themselves cannot provide. However, in defining these principles, it is all too easy to provide general and abstract definitions that are not very helpful to the designer. Therefore, the future of interactive system design relies on a complementary approach. The creativity that gives rise to new paradigms should be strengthened by the development of a theory that provides principles to support the paradigm in its repeated application. We will consider such principles and design rules in detail in Chapter 7 and more theoretical perspectives in Part 3.

EXERCISES

4.1. Choose one of the people mentioned in this chapter, or another important figure in the history of HCI, and create a web page biography on them. Try to get at least one picture of your subject, and find out about their life and work, with particular reference to their contribution to HCI.

4.2. Choose one paradigm of interaction and find three specific examples of it, not included in this chapter. Compare these three – can you identify any general principles of interaction that are embodied in each of your examples (see Chapter 7 for example principles)?

4.3. What new paradigms do you think may be significant in the future of interactive computing?

4.4. A truly ubiquitous computing experience would require the spread of computational capabilities literally everywhere. Another way to achieve ubiquity is to carry all of your computational needs with you everywhere, all the time. The field of *wearable computing* explores this interaction paradigm. How do you think the first-person emphasis of wearable computing compares with the third-person, or environmental, emphasis of ubiquitous computing? What impact would there be on context-aware computing if all of the sensors were attached to the individual instead of embedded in the environment?

RECOMMENDED READING

H. Rheingold, *Tools for Thought*, Prentice Hall, 1985.
An easy to read history of computing, with particular emphasis on developments in interactive systems. Much of the historical perspective of this chapter was influenced by this book.

T. Berners-Lee, *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web*, Harper, 2000.
The history of the world wide web as told by its inventor Tim Berners-Lee makes interesting and enlightening reading.

M. M. Waldrop, *The Dream Machine: J.C.R. Licklider and the Revolution That Made Computing Personal*, Penguin, 2002.
A readable biography of Licklider and his pioneering work in making computing available to all.

T. Bardini, *Bootstrapping: Douglas Englebart, Coevolution and the Origins of Personal Computing (Writing Science)*, Stanford University Press, 2000.
Biography of another great pioneer of interactive computing, Doug Engelbart and his work at Stanford.

J. M. Nyce and P. Kahn, editors, and V. Bush, *From Memex to Hypertext: Vannevar Bush and the Mind's Machine*, Academic Press, 1992.
An edited collection of Bush's writing about Memex, together with essays from computing historians.

M. Weiser, Some computer science issues in ubiquitous computing, *Communications of the ACM*, Vol. 36, No. 7, pp. 75–84, July 1993.
Classic article on issues relating to ubiquitous computing, including hardware, interaction software, networks, applications and methods.

A. Dix, T. Rodden, N. Davies, J. Trevor, A. Friday and K. Palfreyman, Exploiting space and location as a design framework for interactive mobile systems, *ACM Transactions on Computer–Human Interaction (TOCHI)*, Vol. 7, No. 3, pp. 285–321, September 2000.
Explores the space of context sensitive designs focussed especially on location.

There are also a number of special journal issues on ubiquitous, sensor-based and context-aware computing in *ACM TOCHI*, *Human–Computer Interaction* and *Communications of the ACM* and dedicated journals such as *IEEE Pervasive* and *Personal Technologies Journal*.

PART

2

Design process

In this part, we concentrate on how design practice addresses the critical feature of an interactive system – usability from the human perspective. The chapters in this part promote the purposeful design of more usable interactive systems. We begin in Chapter 5 by introducing the key elements in the interaction design process. These elements are then expanded in later chapters.

Chapter 6 discusses the design process in more detail, specifically focussing on the place of user-centered design within a software engineering framework. Chapter 7 highlights the range of design rules that can help us to specify usable interactive systems, including abstract principles, guidelines and other design representations.

In Chapter 8, we provide an overview of implementation support for the programmer of an interactive system. Chapter 9 is concerned with the techniques used to evaluate the interactive system to see if it satisfies user needs. Chapter 10 discusses how to design a system to be universally accessible, regardless of age, gender, cultural background or ability. In Chapter 11 we discuss the provision of user support in the form of help systems and documentation.

5 INTERACTION DESIGN BASICS

OVERVIEW

Interaction design is about creating interventions in often complex situations using technology of many kinds including PC software, the web and physical devices.

- Design involves:
  - achieving goals within constraints and trade-off between these
  - understanding the raw materials: computer and human
  - accepting limitations of humans and of design.
- The design process has several stages and is iterative and never complete.
- Interaction starts with getting to know the users and their context:
  - finding out who they are and what they are like . . . probably not like you!
  - talking to them, watching them.
- Scenarios are rich design stories, which can be used and reused throughout design:
  - they help us see what users will want to do
  - they give a step-by-step walkthrough of users' interactions: including what they see, do and are thinking.
- Users need to find their way around a system. This involves:
  - helping users know where they are, where they have been and what they can do next
  - creating overall structures that are easy to understand and fit the users' needs
  - designing comprehensible screens and control panels.
- Complexity of design means we don't get it right first time:
  - so we need iteration and prototypes to try out and evaluate
  - but iteration can get trapped in *local maxima*, designs that have no simple improvements, but are not good
  - theory and models can help give good start points.

5.1 INTRODUCTION

Some of HCI is focussed on understanding: the academic study of the way people interact with technology. However, a large part of HCI is about doing things and making things – design.

In this chapter we will think about interaction design. Note that we are not just thinking about the design of interactive systems, but about the interaction itself. An office has just got a new electric stapler. It is connected to the mains electricity and is hard to move around, so when you want to staple papers together you go to the stapler. In the past when someone wanted to staple things they would take the stapler to their desk and keep it until someone else wanted it. You might write a letter, print it, staple it, write the next letter, staple it, and so on. Now you have to take the letters to be stapled across the office, so instead you write–print, write–print until you have a pile of things to staple and then take them across. The stapler influences the whole pattern of interaction.

So, interaction design is not just about the artifact that is produced, whether a physical device or a computer program, but about understanding and choosing how that is going to affect the way people work. Furthermore, the artifacts we give to people are not just these devices and programs, but also manuals, tutorials, online help systems. In some cases we may realize that no additional system is required at all, we may simply suggest a different way of using existing tools.

Because of this it may be better not to think of designing a system, or an artifact, but to think instead about *designing interventions*. The product of a design exercise is that we intervene to change the situation as it is; we hope, of course, changing it for the better!

In the next section we will ask 'what is design?' which sets the spirit for the rest of the chapter. Section 5.3 looks at the design process as a whole and this gives a framework for the following sections. Section 5.4 looks at aspects of the requirements-gathering phase of design focussed on getting to know and understand the user. This is followed in Section 5.5 by a look at scenarios, which are a way of recording existing situations and examining proposed designs. We then look at the details of designing the overall application structure in Section 5.6 and individual screen design in Section 5.7. Because design is never perfect first time (or ever!), most interaction design involves several cycles of prototyping and evaluation. The chapter ends with an examination of the limits of this and why this emphasizes the importance of deep knowledge of more general theories and models of interaction.

This chapter also functions as an introduction to much of Part 2 and Part 3 of this book. In particular, Section 5.3 puts many of the succeeding chapters into the context of the overall design process. Many of the individual sections of this chapter give early views, or simple techniques, for issues and areas dealt with in detail later in the book.

5.2 WHAT IS DESIGN?

So what is design? A simple definition is:

> achieving goals within constraints

This does not capture everything about design, but helps to focus us on certain things:

Goals What is the purpose of the design we are intending to produce? Who is it for? Why do they want it? For example, if we are designing a wireless personal movie player, we may think about young affluent users wanting to watch the latest movies whilst on the move and download free copies, and perhaps wanting to share the experience with a few friends.

Constraints What materials must we use? What standards must we adopt? How much can it cost? How much time do we have to develop it? Are there health and safety issues? In the case of the personal movie player: does it have to withstand rain? Must we use existing video standards to download movies? Do we need to build in copyright protection?

Of course, we cannot always achieve all our goals within the constraints. So perhaps one of the most important things about design is:

Trade-off Choosing which goals or constraints can be relaxed so that others can be met. For example, we might find that an eye-mounted video display, a bit like those used in virtual reality, would give the most stable image whilst walking along. However, this would not allow you to show friends, and might be dangerous if you were watching a gripping part of the movie as you crossed the road.

Often the most exciting moments in design are when you get a radically different idea that allows you to satisfy several apparently contradictory constraints. However, the more common skill needed in design is to accept the conflict and choose the most appropriate trade-off. The temptation is to focus on one or other goal and optimize for this, then tweak the design to make it just satisfy the constraints and other goals. Instead, the best designs are where the designer understands the trade-offs and the factors affecting them. Paradoxically, if you focus on the trade-off itself the more radical solutions also become more apparent.

5.2.1 The golden rule of design

Part of the understanding we need is about the circumstances and context of the particular design problem. We will return to this later in the chapter. However, there are also more generic concepts to understand. The designs we produce may be different, but often the raw materials are the same. This leads us to the *golden rule of design*:

> understand your materials

In the case of a physical design this is obvious. Look at a chair with a steel frame and one with a wooden frame. They are very different: often the steel frames are tubular or thin L or H section steel. In contrast wooden chairs have thicker solid legs. If you made a wooden chair using the design for a metal one it would break; if you made the metal one in the design for the wooden one it would be too heavy to move.

For Human–Computer Interaction the obvious materials are the human and the computer. That is we must:

- understand *computers*
  - limitations, capacities, tools, platforms
- understand *people*
  - psychological, social aspects, human error.

Of course, this is exactly the focus of Chapters 1 and 2. This is why they came first; we must understand the fundamental materials of human–computer interaction in order to design. In Chapters 3 and 4 we also looked at the nature of *interaction* itself. This is equally important in other design areas. For example, the way you fit seats and windows into an airplane's hull affects the safety and strength of the aircraft as a whole.

5.2.2 To err is human

It might sound demeaning to regard people as 'materials', possibly even dehumanizing. In fact, the opposite is the case: physical materials are treated better in most designs than people. This is particularly obvious when it comes to failures.

The news headlines: an aircrash claims a hundred lives; an industrial accident causes millions of pounds' worth of damage; the discovery of systematic mistreatment leads to thousands of patients being recalled to hospital. Some months later the public inquiries conclude: human error in the operation of technical instruments. The phrase 'human error' is taken to mean 'operator error', but more often than not the disaster is inherent in the design or installation of the human interface. Bad interfaces are slow or error-prone to use. Bad interfaces cost money and cost lives.

People make mistakes. This is not 'human error', an excuse to hide behind in accident reports, it is human nature. We are not infallible consistent creatures, but often make slips, errors and omissions. A concrete lintel breaks and a building collapses. Do the headlines read 'lintel error'? No. It is the nature of concrete lintels to break if they are put under stress and it is the responsibility of architect and engineer to ensure that a building only puts acceptable stress on the lintel. Similarly it is the nature of humans to make mistakes, and systems should be designed to reduce the likelihood of those mistakes and to minimize the consequences when mistakes happen.

Often when an aspect of an interface is obscure and unclear, the response is to add another line in the manual. People are remarkably adaptable and, unlike concrete lintels, can get 'stronger', but better training and documentation (although necessary) are not a panacea. Under stress, arcane or inconsistent interfaces will lead to errors.

If you design using a physical material, you need to understand how and where failures would occur and strengthen the construction, build in safety features or redundancy. Similarly, if you treat the human with as much consideration as a piece of steel or concrete, it is obvious that you need to understand the way human failures occur and build the rest of the interface accordingly.

5.2.3 The central message – the user

In this book you will find information on basic psychology, on particular technologies, on methods and models. However, there is one factor that outweighs all this knowledge. It is about attitude. Often it is said that the success of the various methods used in HCI lies not in how good they are, but in that they simply focus the mind of the designer on the user.

This is the core of interaction design: put the user first, keep the user in the center and remember the user at the end.

5.3 THE PROCESS OF DESIGN

Often HCI professionals complain that they are called in too late. A system has been designed and built, and only when it proves unusable do they think to ask how to do it right! In other companies usability is seen as equivalent to testing – checking whether people can use it and fixing problems, rather than making sure they can from the beginning. In the best companies, however, usability is designed in from the start.

In Chapter 6 we will look in detail at the software development process and how HCI fits within it. Here we'll take a simplified view of four main phases plus an iteration loop, focussed on the design of interaction (Figure 5.1).

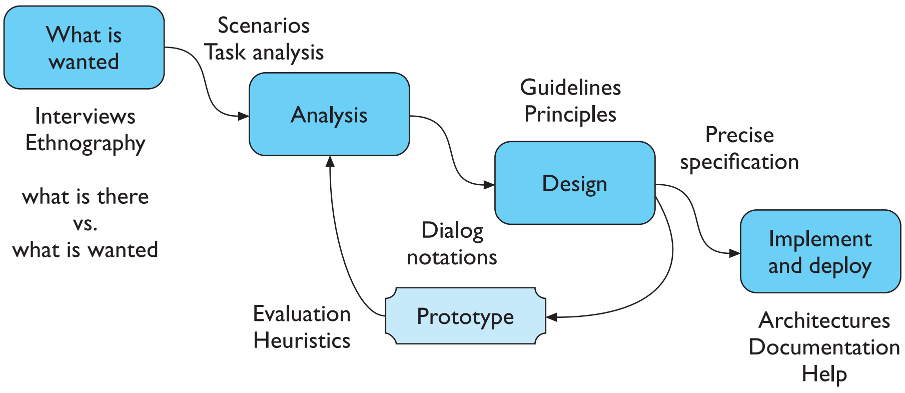


Figure 5.1 Interaction design process

Requirements – what is wanted The first stage is establishing what exactly is needed. As a precursor to this it is usually necessary to find out what is currently happening. For example, how do people currently watch movies? What sort of personal appliances do they currently use?

There are a number of techniques used for this in HCI: interviewing people, videotaping them, looking at the documents and objects that they work with, observing them directly. We don't have a chapter dedicated to this, but aspects will be found in various places throughout the book. In particular, ethnography, a form of observation deriving from anthropology, has become very influential and is discussed in Chapter 13. We will look at some ways of addressing this stage in Section 5.4.

Analysis The results of observation and interview need to be ordered in some way to bring out key issues and communicate with later stages of design. Chapter 15 and part of Chapter 18 deal with task models, which are a means to capture how people carry out the various tasks that are part of their work and life. In this chapter (Section 5.5), we will look at scenarios, rich stories of interaction, which can be used in conjunction with a method like task analysis or on their own to record and make vivid actual interaction. These techniques can be used both to represent the situation as it is and also the desired situation.

Design Well, this is all about design, but there is a central stage when you move from what you want, to how to do it. There are numerous rules, guidelines and design principles that can be used to help with this and Chapter 7 discusses these in detail; whilst Chapter 10 looks at how to design taking into account many different kinds of user. We need to record our design choices in some way and there are various notations and methods to do this, including those used to record the existing situation. Chapters 16, 17 and 18 deal with ways of modeling and describing interaction. In this chapter, Section 5.6 will look at some simple notations for designing navigation within a system and some basic heuristics to guide the design of that navigation. Section 5.7 will look more closely at the layout of individual screens. It is at this stage also where input from theoretical work is most helpful, including cognitive models, organizational issues and understanding communication (Chapters 12, 13 and 14).

Iteration and prototyping Humans are complex and we cannot expect to get designs right first time. We therefore need to evaluate a design to see how well it is working and where there can be improvements. We will discuss some techniques for evaluation in Chapter 9. Some forms of evaluation can be done using the design on paper, but it is hard to get real feedback without trying it out. Most user interface design therefore involves some form of prototyping, producing early versions of systems to try out with real users. We'll discuss this in Section 5.8.

Implementation and deployment Finally, when we are happy with our design, we need to create it and deploy it. This will involve writing code, perhaps making hardware, writing documentation and manuals – everything that goes into a real system that can be given to others. Chapter 8 will deal with software architectures for user interfaces and there are details about implementing groupware in Chapter 19 and web interfaces in Chapter 21.

If you read all the chapters and look at all the techniques you might think 'help! how can I ever do all this?'. Of course the answer is you can't. Your time is limited – there is a trade-off between the length of the design period and the quality of the final design. This means one sometimes has to accept a design as final even if it is not perfect: it is often better to have a product that is acceptable but on time and to cost than it is to have one that has perfect interaction but is late and over budget.

It is easy to think that the goal, especially of the iterative stages, is to find usability problems and fix them. As you experience real designs, however, you soon find that the real problem is not to find faults – that is easy; nor to work out how to fix them – that may not be too difficult; instead the issue is: which usability problems is it worth fixing?

In fact, if you ever come across a system that seems to be perfect it is a badly designed system – badly designed not because the design is bad, but because too much effort will have been spent in the design process itself. Just as with all trade-offs, it may be possible to find radically different solutions that have a major effect but are cheap to implement. However, it is best not to plan assuming such bolts of inspiration will strike when wanted!

5.4 USER FOCUS

As we've already said, the start of any interaction design exercise must be the intended user or users. This is often stated as:

know your users

Because this sounds somewhat like a commandment it is sometimes even written 'know thy user' (and originally 'know the user' [162]). Note, too, a little indecision about user/users – much of traditional user interface design has focussed on a single user. We will discuss issues of collaboration extensively in Chapters 13 and 19, but even at this stage it is important to be aware that there is rarely one user of a system. This doesn't mean that every system is explicitly supporting collaboration like email does. However, almost every system has an impact beyond the person immediately using it.

Think about a stock control system. The warehouse manager queries the system to find out how many six-inch nails are in stock – just a single user? Why did he do this? Perhaps a salesperson has been asked to deliver 100,000 six-inch nails within a fortnight and wants to know if the company is able to fulfill the order in time. So the act of looking at the stock control system involves the warehouse manager, the salesperson and the client. The auditors want to produce a valuation of company assets including stock in hand, the assistant warehouse manager needs to update the stock levels while his boss is on holiday.

Over time many people are affected directly or indirectly by a system and these people are called *stakeholders* (see also Chapter 13). Obviously, tracing the tenuous links between people could go on for ever and you need to draw boundaries as to whom you should consider. This depends very much on the nature of the systems being designed, but largely requires plain common sense.

So, how do you get to know your users?

■ Who are they?

Of course, the first thing to find out is who your users are. Are they young or old, experienced computer users or novices? As we saw with the stock control system, it may not be obvious who the users are, so you may need to ask this question again as you find out more about the system and its context. This question becomes harder to answer if you are designing generic software, such as a word processor, as there are many different users with different purposes and characteristics. A similar problem arises with many websites where the potential visitors are far from homogenous. It may be tempting to try to think of a generic user with generic skills and generic goals; however, it is probably better, either instead or in addition, to think of several specific users.

■ Probably *not* like you!

When designing a system it is easy to design it as if *you* were the main user: you assume your own interests and abilities. So often you hear a designer say 'but it's obvious what to do'. It may be obvious for her! This is not helped by the fact that many software houses are primarily filled with male developers. Although individuals differ a lot there is a tendency for women to have better empathetic skills.

■ Talk to them.

It is hard to get yourself inside someone else's head, so the best thing is usually to ask them. This can take many forms: structured interviews about their job or life, open-ended discussions, or bringing the potential users fully into the design process. The last of these is called *participatory design* (see Chapter 13, Section 13.3.4). By involving users throughout the design process it is possible to get a deep knowledge of their work context and needs. The obvious effect of this is that it produces better designs. However, there is a second motivational effect, perhaps at least as important as the quality of the design. By being involved, users come to 'own' the design and become champions for it once deployed. Recall that a system must be not only useful and usable, but also *used*.

People may also be able to tell you about how things *really* happen, not just how the organization says they *should* happen. To encourage users to tell you this, you will need to win their trust, since often the actual practices run counter to corporate policy. However it is typically these ad hoc methods that make organizations work, not the official story!

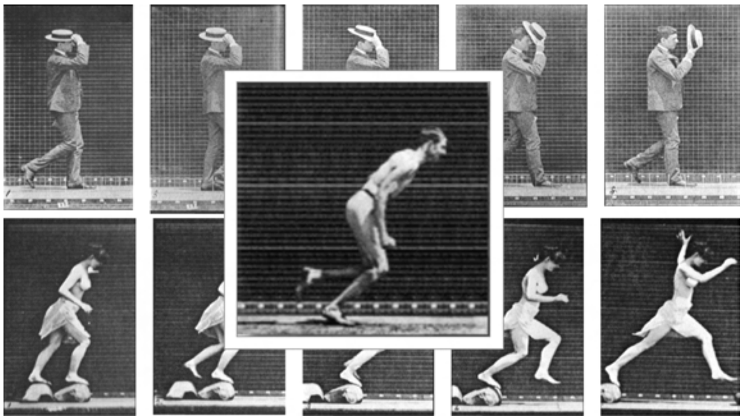


Figure 5.2 Eadweard Muybridge's time-lapse photography. Source for top plate and middle plate: Kingston Museum and Heritage Service; source for bottom plate: V&A Images, The Victoria and Albert Museum, London

- Watch them.

Although what people tell you is of the utmost importance, it is not the whole story.

When black-belt judo players are asked how they throw an opponent, their explanations do not match what they actually do. Think about walking – how do your legs and arms move? It is harder than you would think! Although people have run since the earliest times, it was only with Eadweard Muybridge's pioneering time-lapse photography in the 1870s that the way people actually walk, run and move became clear (see Figure 5.2). This is even more problematic with intellectual activities as it is notoriously difficult to introspect.

A professional in any field is very practiced and can *do* things in the domain. An academic in the same field may not be able to do things, but she knows *about* the things in the domain. These are different kinds of knowledge and skill. Sometimes people know both, but not necessarily so. The best sports trainers may not be the best athletes, the best painters may not be the best art critics.

Because of this it is important to watch what people do as well as hear what they say. This may involve sitting and taking notes of how they spend a day, watching particular activities, using a video camera or tape recorder. It can be done in an informal manner or using developed methods such as ethnography or contextual inquiry, which we will discuss in Chapter 13.

Sometimes users can be involved in this; for example, asking them to keep a diary or having a 15-minute buzzer and asking them to write down what they are doing when the buzzer sounds. Although this sounds just like asking the users what they do, the structured format helps them give a more accurate answer.

Another way to find out what people are doing is to look at the artifacts they are using and creating. Look at a typical desk in an office. There are papers, letters, files, perhaps a stapler, a computer, sticky notes . . . Some of these carry information, but if they were only important for the information in them they could equally well be

Design Focus

Cultural probes

Traditional ethnography has involved watching people and being present. There is always a disruptive effect when someone is watching, but in, say, an office, after a while the ethnographer becomes 'part of the wallpaper' and most activities carry on as normal. However, in some environments, for example the home or with psychiatric patients, it is hard to go and watch people for long periods if at all. Cultural probes have been used as one way to gather rich views of an area without intrusion. These were originally developed as prompts for design [146], but have also been adopted as an added method for ethnography [170].

Source: Photograph courtesy of William W. Gaver

Cultural probes are small packs of items designed to provoke and record comments in various ways. They are given to people to take away and to open and use in their own environment. For example, one probe pack for the domestic environment includes a glass with a paper sleeve. You use the glass to listen to things and then write down what you hear. The same probe pack contains a repackaged disposable camera and a small solid-state voice recorder. When the packs are returned, the notes, recordings, photos, etc., are used as a means of understanding what is significant and important for the people in the environment and as a means of enculturing designers.

For more see /e3/online/cultural-probes/ and www.crd.rca.ac.uk/equator/domestic_probes.html

in the filing cabinet and just taken out when needed. The sticky note on the edge of Brian's screen saying 'book table' is not just information that he needs to book a restaurant table. The fact that it is on his screen is reminding him that something needs to be done. In Chapter 18 we will look at the role of artifacts in detail.

Betty is 37 years old. She has been Warehouse Manager for five years and has worked for Simpkins Brothers Engineering for 12 years. She didn't go to university, but has studied in her evenings for a business diploma. She has two children aged 15 and 7 and does not like to work late. She did part of an introductory in-house computer course some years ago, but it was interrupted when she was promoted and could no longer afford to take the time. Her vision is perfect, but her right-hand movement is slightly restricted following an industrial accident three years ago. She is enthusiastic about her work and is happy to delegate responsibility and take suggestions from her staff. However, she does feel threatened by the introduction of yet another new computer system (the third in her time at SBE).

Figure 5.3 Persona – a rich description of Betty the Warehouse Manager

In all these observational methods one should not just stop at the observation, but go back and discuss the observations with the users. Even if they were not previously aware of what they were doing, they are likely to be able to explain when shown. The observations tell you *what* they do, they will tell you *why*.

- Use your imagination.

Even if you would like to involve many users throughout your design exercise this will not always be possible. It may be too costly, it may be hard to get time with them (e.g. hospital consultant), it may be that there are just too many (e.g. the web). However, even if you cannot involve actual users you can at least try to imagine their experiences.

Now this is very dangerous! It would be easy to think, 'if I were a warehouse manager I would do this'. The issue is not what *you* would do in the user's shoes but what *they* would do. This requires almost a kind of method acting. Imagine being a warehouse manager. What does the word 'undo' in the menu mean to him?

One method that has been quite successful in helping design teams produce user-focussed designs is the *persona*. A persona is a rich picture of an imaginary person who represents your core user group. Figure 5.3 gives an example persona of Betty the warehouse manager. A design team will have several of these personae covering different types of intended users and different roles. The personae will themselves be based on studies of actual users, observation, etc. When a design solution is proposed the team can ask, 'how would Betty react to this?'. The detail is deliberately more than is strictly necessary, but this is essential. It is only by feeling that Betty is a real person that the team can start to imagine how she will behave.

5.5 SCENARIOS

Scenarios are stories for design: rich stories of interaction. They are perhaps the simplest design representation, but one of the most flexible and powerful. Some scenarios are quite short: 'the user intends to press the "save" button, but accidentally

Brian would like to see the new film *Moments of Significance* and wants to invite Alison, but he knows she doesn't like 'arty' films. He decides to take a look at it to see if she would like it and so connects to one of the movie-sharing networks. He uses his work machine as it has a higher bandwidth connection, but feels a bit guilty. He knows he will be getting an illegal copy of the film, but decides it is OK as he is intending to go to the cinema to watch it. After it downloads to his machine he takes out his new personal movie player. He presses the 'menu' button and on the small LCD screen he scrolls using the arrow keys to 'bluetooth connect' and presses the 'select' button. On his computer the movie download program now has an icon showing that it has recognized a compatible device and he drags the icon of the film over the icon for the player. On the player the LCD screen says 'downloading now', with a per cent done indicator and small whirling icon.

During lunchtime Brian takes out his movie player, plugs in his earphones and starts to watch. He uses the arrow keys to skip between portions of the film and decides that, yes, Alison would like it. Then he feels a tap on his shoulder. He turns round. It is Alison. He had been so absorbed he hadn't noticed her. 'What are you watching', she says. 'Here, listen', he says and flicks a small switch. The built-in directional speaker is loud enough for both Brian and Alison to hear, but not loud enough to disturb other people in the canteen. Alison recognizes the film from trailers, 'surprised this is out yet' she says. 'Well actually . . .', Brian confesses, 'you'd better come with me to see it and make an honest man of me'. 'I'll think about it', she replies.

Figure 5.4 Scenario for proposed movie player

presses the "quit" button so loses his work'. Others are focussed more on describing the situation or context.

Figure 5.4 gives an example of a scenario for the personal movie player. Like the persona it is perhaps more detailed than appears necessary, but the detail helps make the events seem real. The figure shows plain text, but scenarios can be augmented by sketches, simulated screen shots, etc. These sketches and pictures are called *storyboards* and are similar to the techniques used in film making to envisage plot-lines.

Where the design includes physical artifacts the scenarios can be used as a script to act out potential patterns of use. For example, we might imagine a digital Swiss army knife, which has a small LCD screen and uses the toothpick as a stylus. The knife connects to the internet via a wireless link through your phone and gives interesting tips from other Swiss army knife users. Try getting two together at a party – you will see this would appeal! It sounds like a great design idea – but wait, try acting out the use. If you have a Swiss army knife, use it, or use something penknife-sized if you don't. The tip on the LCD says, 'open the stone remover': a small LED glows near the right blade – you open it. 'Now push the blade into the rubber of the grommet', it says. You do this and then look for the next instruction. Look at the knife in your hand . . . oops, your thumb is covering where the screen would be. Perhaps a voice interface would be better.

You can see already how scenarios force you to think about the design in detail and notice potential problems before they happen. If you add more detail you can get to a blow-by-blow account of the user–system interactions and then ask 'what is the user intending now?'; 'what is the system doing now?'. This can help to verify that the design would make sense to the user and also that proposed implementation architectures would work.

In addition scenarios can be used to:

Communicate with others – other designers, clients or users. It is easy to misunderstand each other whilst discussing abstract ideas. Concrete examples of use are far easier to share.

Validate other models A detailed scenario can be 'played' against various more formal representations such as task models (discussed in Chapter 15) or dialog and navigation models (Chapter 16 and below).

Express dynamics Individual screen shots and pictures give you a sense of what a system would look like, but not how it behaves.

In the next section we will discuss ways of describing the patterns of interaction with a system. These are more complex and involve networks or hierarchies. In contrast scenarios are linear – they represent a single path amongst all the potential interactions.

This linearity has both positive and negative points:

Time is linear Our lives are linear as we live in time and so we find it easier to understand simple linear narratives. We are natural storytellers and story listeners.

But no alternatives Real interactions have choices, some made by people, some by systems. A simple scenario does not show these alternative paths. In particular, it is easy to miss the unintended things a person may do.

Scenarios are a resource that can be used and reused throughout the design process: helping us see what is wanted, suggesting how users will deal with the potential design, checking that proposed implementations will work, and generating test cases for final evaluation.

For more examples of scenarios see: /e3/online/scenario/

5.6 NAVIGATION DESIGN

As we stressed, the object of design is not just a computer system or device, but the socio-technical intervention as a whole. However, as design progresses we come to a point where we do need to consider these most tangible outputs of design.

Imagine yourself using a word processor. You will be doing this in some particular social and physical setting, for a purpose. But now we are focussing on the computer system itself. You interact at several levels:

Widgets The appropriate choice of widgets and wording in menus and buttons will help you know how to use them for a particular selection or action.

Screens or windows You need to find things on the screen, understand the logical grouping of buttons.

Table 5.1 Levels of interaction

PC application	Website	Physical device
Widgets	Form elements, tags and links	Buttons, dials, lights, displays
Screen design	Page design	Physical layout
Navigation design	Site structure	Main modes of device
Other apps and operating system	The web, browser, external links	The real world!

Navigation within the application You need to be able to understand what will happen when a button is pressed, to understand where you are in the interaction.

Environment The word processor has to read documents from disk, perhaps some are on remote networks. You swap between applications, perhaps cut and paste.

You can see similar levels in other types of application and device, as Table 5.1 shows. There are differences; for example, in the web we have less control of how people enter a site and on a physical device we have the same layout of buttons and displays no matter what the internal state (although we may treat them differently).

We discussed graphical user interface widgets in Chapter 3 and in the next section we will look at details of screen design. In this section we will look mainly at navigation design, that is the main screens or modes within a system and how they interconnect. We will also briefly consider how this interacts with the wider environment.

Just in case you haven't already got the idea, the place to start when considering the structure of an application is to think about actual use:

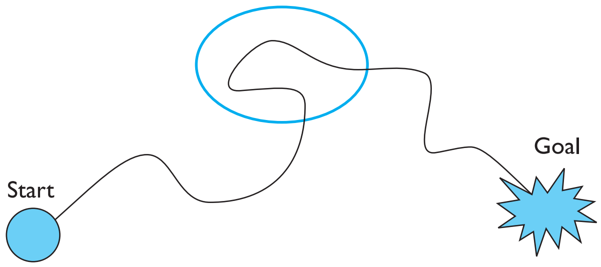
- who is going to use the application?
- how do they think about it?
- what will they do with it?

This can then drive the second task – thinking about structure. Individual screens or the layout of devices will have their own structure, but this is for the next section. Here we will consider two main kinds of issue:

- local structure
  - looking from one screen or page out
- global structure
  - structure of site, movement between screens.

5.6.1 Local structure

Much of interaction involves goal-seeking behavior. Users have some idea of what they are after and a partial model of the system. In an ideal world if users had perfect knowledge of what they wanted and how the system worked they could simply take the shortest path to what they want, pressing all the right buttons and links. However, in a world of partial knowledge users meander through the system. The important thing is not so much that they take the most efficient route, but that at each point in the interaction they can make some assessment of whether they are getting closer to their (often partially formed) goal.



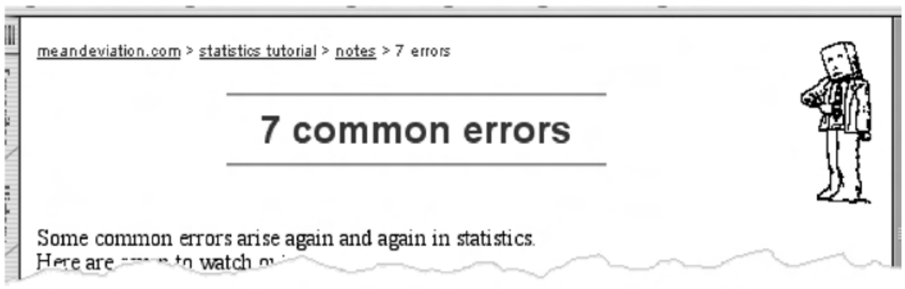
To do this goal seeking, each state of the system or each screen needs to give the user enough knowledge of what to do to get closer to their goal. In Chapter 7 we will look at various design rules, some of which address this issue. To get you started, here are four things to look for when looking at a single web page, screen or state of a device.

- knowing where you are
- knowing what you can do
- knowing where you are going – or what will happen
- knowing where you've been – or what you've done.

The screen, web page or device displays should make clear *where you are* in terms of the interaction or state of the system. Some websites show 'bread crumbs' at the top of the screen, the path of titles showing where the page is in the site (Figure 5.5). Similarly, in the scenario in Figure 5.4, the personal movie player says 'downloading now', so Brian knows that it is in the middle of downloading a movie from the PC.

It is also important to know *what you can do* – what can be pressed or clicked to go somewhere or do something. Some web pages are particularly bad in that it is unclear which images are pure decoration and which are links to take you somewhere.

On the web the standard underlined links make it clear which text is clickable and which is not. However, in order to improve the appearance of the page many sites change the color of links and may remove the underline too. This is especially confusing if underline is then used as simple emphasis on words that are not links! The


meandeviation.com > statistics tutorial > notes > 7 errors

7 common errors

Some common errors arise again and again in statistics.
Here are

Figure 5.5 Breadcrumbs. Screen shot frame reprinted by permission from Microsoft Corporation

trade-off between appearance and ease of use may mean that this is the right thing to do, but you should take care before confusing the user needlessly.

Chic design is also a problem in physical devices. One of the authors was once in a really high-class hotel and found he could not locate the flush in the toilet. Only after much fumbling did he discover that one of the tiles could be pressed. The ‘active’ tile was level with the rest of the tiled wall – a very clean design, but not very usable!

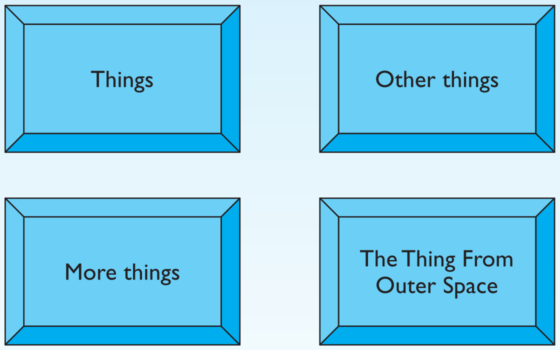
You then need to know *where you are going* when you click a button or *what will happen*. Of course you can try clicking the button to see. In the case of a website or information system this may mean you then have to use some sort of ‘back’ mechanism to return, but that is all; however, in an application or device the action of clicking the button may already have caused some effect. If the system has an easy means to undo or reverse actions this is not so bad, but it is better if users do not have to use this ‘try it and see’ interaction. Where response times are slow this is particularly annoying.

Remember too that icons are typically not self-explanatory and should always be accompanied by labels or at the very least tooltips or some similar technique. A picture paints a thousand words, but typically only when explained first using fifteen hundred!

Design Focus

Beware the big button trap

Public information systems often have touchscreens and so have large buttons. Watch someone using one of these and see how often they go to the wrong screen and have to use ‘back’ or ‘home’ to try again. If you look more closely you will find that each button has only one or two words on it giving the title of the next screen, and possibly some sort of icon. Quite rightly, the button label will be in a large font as users may have poor eyesight.



It is hard to choose appropriate labels that mean the same for everyone, especially when the breadth of the screen hierarchy is fixed by the maximum number of buttons. So it is no wonder that people get confused. However, there is usually plenty of room for additional explanation in a smaller font, possibly just the next level of button labels, or a sentence of explanation. It may not look as pretty, but it may mean that people actually find the information they are looking for.

Special care has to be taken if the same command or button press means something different in different contexts. These different contexts that change the interpretation of commands are called *modes*. Many older text editors would interpret pressing 'x' to mean 'enter me into the text' in a normal typing mode, but 'exit' in a special command mode. If modes are clearly visible or audible this is less of a problem and in Chapter 3 (Section 3.6.7) we saw how palettes are one way to achieve this. In general, modes are less of a problem in windowed systems where the mode is made apparent by the current window (if you remember which it is). However, physical devices may have minimal displays and may be operated without visual attention.

Finally, if you have just done some major action you also want some sort of confirmation of *what you've done*. If you are faultless and have perfect knowledge, of course you will be sure that you have hit the right key and know exactly what

Design Focus

Modes

Alan's mobile phone has a lock feature to prevent accidental use. To remove the lock he has to press the 'C' (cancel) button which then asks for an additional 'yes' to confirm removing the lock. So, in 'locked' mode, 'C' followed by 'yes' means 'turn off lock' and these are the most frequent actions when Alan takes the phone from his pocket.

However, Alan is forgetful and sometimes puts the phone in his pocket unlocked. This leads to occasional embarrassing phone calls and also to another problem.

The 'yes' button is quite big and so this is often pressed while in his pocket. This puts the phone into 'dial recent numbers' mode with a list of recent calls on screen. In this mode, pressing 'C' gives a prompt 'delete number' and pressing 'yes' then deletes the number from the phone's address book. Unhappily, this often means he takes the phone from his pocket, automatically presses 'C', 'yes' only to see as he looks down to the handset the fatal words 'number deleted'. Of course there is no undo!

will happen. Remember, too, that to know what will happen, you would need to know everything about the internal state of the system and things outside, like the contents of files, networked devices, etc., that could affect it. In other words, if you were omniscient you could do it. For lesser mortals the system needs to give some *feedback* to say what has happened.

In an information system, there is a related but slightly different issue, which is to know *where you have been*. This helps you to feel in control and understand your navigation of the information space. The feeling of disorientation when you do not have sufficient means to know where you are and where you have been has been called 'lost in hyperspace'. Most web browsers offer a history system and also a 'back' button that keeps a list of recently visited pages.

5.6.2 Global structure – hierarchical organization

We will now look at the overall structure of an application. This is the way the various screens, pages or device states link to one another.

One way to organize a system is in some form of hierarchy. This is typically organized along functional boundaries (that is, different kinds of things), but may be organized by roles, user type, or some more esoteric breakdown such as modules in an educational system.

The hierarchy links screens, pages or states in logical groupings. For example, Figure 5.6 gives a high-level breakdown of some sort of messaging system. This sort of hierarchy can be used purely to help during design, but can also be used to structure the actual system. For example, this may reflect the menu structure of a PC application or the site structure on the web.

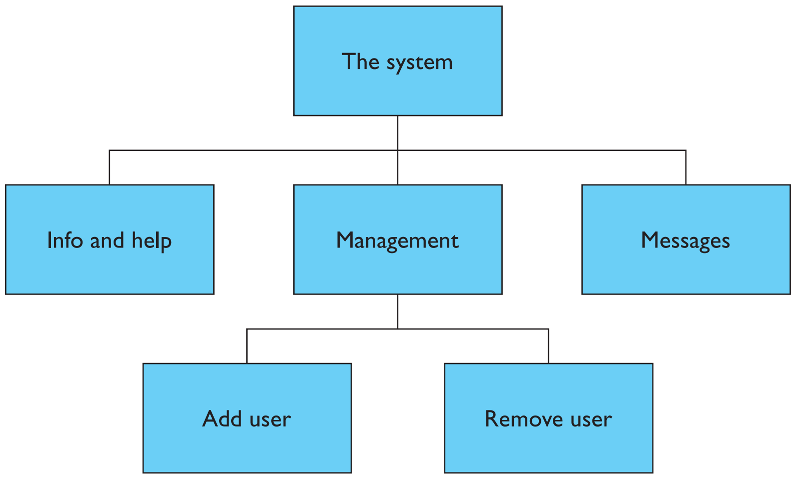


Figure 5.6 Application functional hierarchy

Any sort of information structuring is difficult, but there is evidence that people find hierarchies simpler than most. One of the difficulties with organizing information or system functionality is that different people have different internal structures for their knowledge, and may use different vocabulary. This is one of the places where a detailed knowledge of the intended users is essential: it is no good creating a hierarchy that the designers understand, but not the users . . . and all too commonly this is exactly what happens.

However much you think you have got the wording and categories right, because there are different users it is inevitable that not everyone will understand it perfectly. This is where clear guidance as suggested in Section 5.6.1 (knowing where you are going – or what will happen) is essential, as well as the means to allow users to change their mind if they make the wrong decisions.

There is also evidence that deep hierarchies are difficult to navigate, so it is better to have broad top-level categories, or to present several levels of menu on one screen or web page. Miller's magic number of 7 ± 2 for working memory capacity (see Chapter 1, Section 1.3.2) is often misused in this context. Many guidelines suggest that menu breadth, that is the number of choices available at each level in the menu, should be around seven. However, Miller's result applies only to working memory, not visual search. In fact, optimal breadth can be quite large, perhaps 60 or more items for a web index page if the items are organized in such a way that the eye can easily find the right one [206]. (See /e3/online/menu-breadth/ for more on optimal menu breadth.) Of course, to organize the items on the page requires further classification. However, here the critical thing is the naturalness of the classification, which itself may depend on the user's purpose. For example, if the user wants to look up information on a particular city, an alphabetical list of all city names would be fast, but for other purposes a list by region would be more appropriate.

5.6.3 Global structure – dialog

In a pure information system or static website it may be sufficient to have a fully hierarchical structure, perhaps with next/previous links between items in the same group. However, for any system that involves doing things, constantly drilling down from one part of the hierarchy to another is very frustrating. Usually there are ways of getting more quickly from place to place. For example, in a stock control system there may be a way of going from a stock item to all orders outstanding on that item and then from an order to the purchase record for the customer who placed the order. These would each be in a very different part of a hierarchical view of the application, yet directly accessible from one another.

As well as these cross-links in hierarchies, when you get down to detailed interactions, such as editing or deleting a record, there is obviously a flow of screens and commands that is not about hierarchy. In HCI the word 'dialog' is used to refer to this pattern of interactions between the user and a system.

Consider the following fragment from a marriage service:

Minister:	Do you *name* take this woman . . .
Man:	I do
Minister:	Do you *name* take this man . . .
Woman:	I do
Minister:	I now pronounce you man and wife

Notice this describes the general flow of the service, but has blanks for the names of the bride and groom. So it gives the pattern of the interaction between the parties, but is instantiated differently for each service. Human–computer dialog is just the same; there are overall patterns of movement between main states of a device or windows in a PC application, but the details differ each time it is run.

Recall that scenarios gave just one path through the system. To describe a full system we need to take into account different paths through a system and loops where the system returns to the same screen. There are various ways to do this, and in Chapter 16 we will expand on the wedding example and look at several different types of dialog model.

A simple way is to use a network diagram showing the principal states or screens linked together with arrows. This can:

- show what leads to what
- show what happens when
- include branches and loops
- be more task oriented than a hierarchy.

Figure 5.7 shows a network diagram illustrating the main screens for adding or deleting a user from the messaging system in Figure 5.6. The arrows show the general flow between the states. We can see that from the main screen we can get to either the 'remove user' screen or the 'add user' screen. This is presumably by selecting buttons or links, but the way these are shown we leave to detailed screen design. We can also see that from the 'add user' screen the system always returns to the main screen, but after the 'remove user' screen there is a further confirmation screen.

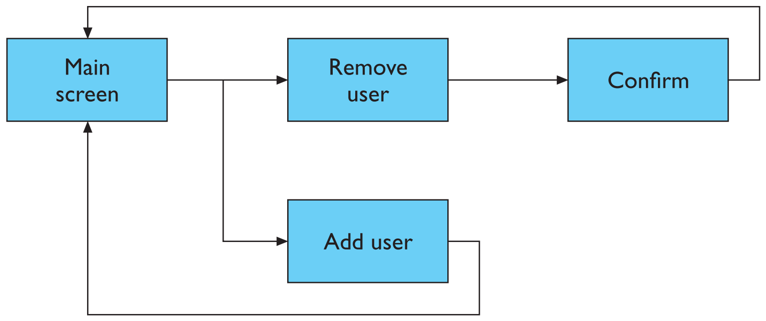


Figure 5.7 Network of screens/states

5.6.4 Wider still

Donne said ‘No man is an Iland, intire of it selfe’. This is also true of the things we design. Each sits amongst other devices and applications and this in turn has to be reflected within our design.

This has several implications:

Style issues We should normally conform to platform standards, such as positions for menus on a PC application, to ensure consistency between applications. For example, on our proposed personal movie player we should make use of standard fast-forward, play and pause icons.

Functional issues On a PC application we need to be able to interact with files, read standard formats and be able to handle cut and paste.

Navigation issues We may need to support linkages between applications, for example allowing the embedding of data from one application in another, or, in a mail system, being able to double click an attachment icon and have the right application launched for the attachment.

On the web we have the added difficulty that other sites and applications may include links that bypass our ‘home page’ and other pages and go direct into the heart of our site or web application. Also, when we link to other sites, we have no control over them or the way their content may change over time.

5.7 SCREEN DESIGN AND LAYOUT

We have talked about the different elements that make up interactive applications, but not about how we put them together. A single screen image often has to present information clearly and also act as the locus for interacting with the system. This is a complex area, involving some of the psychological understanding from Chapter 1 as well as aspects of graphical design.

The basic principles at the screen level reflect those in other areas of interaction design:

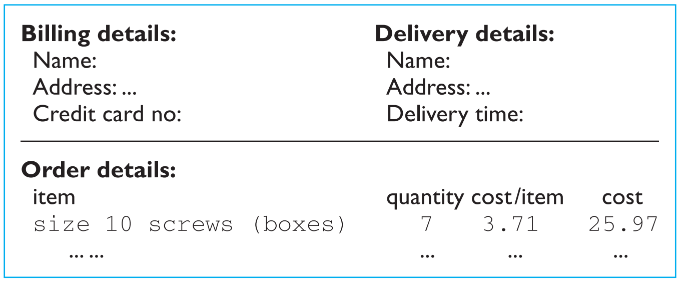
Ask What is the user doing?

Think What information is required? What comparisons may the user need to make? In what order are things likely to be needed?

Design Form follows function: let the required interactions drive the layout.

5.7.1 Tools for layout

We have a number of visual tools available to help us suggest to the user appropriate ways to read and interact with a screen or device.



Billing details:
Name:
Address: ...
Credit card no:

Delivery details:
Name:
Address: ...
Delivery time:

Order details:

item	quantity	cost/item	cost
size 10 screws (boxes)	7	3.71	25.97
... ...	...	...	...

Figure 5.8 Grouping related items in an order screen

Grouping and structure

If things logically belong together, then we should normally physically group them together. This may involve multiple levels of structure. For example, in Figure 5.8 we can see a potential design for an ordering screen. Notice how the details for billing and delivery are grouped together spatially; also note how they are separated from the list of items actually ordered by a line as well as spatially. This reflects the following logical structure:

```
Order:
    Administrative information
        Billing details
        Delivery details
    Order information
        Order line 1
        Order line 2
        ...
```

Order of groups and items

If we look at Figure 5.8 again we can see that the screen seems to naturally suggest reading or filling in the billing details first, followed by the delivery details, followed by the individual order items. Is this the right order?

In general we need to think: what is the natural order for the user? This should normally match the order on screen. For data entry forms or dialog boxes we should also set up the order in which the tab key moves between fields.

Occasionally we may also want to force a particular order; for example we may want to be sure that we do not forget the credit card details!

Decoration

Again looking at Figure 5.8, we can see how the design uses boxes and a separating line to make the grouping clear. Other decorative features like font style, and text or background colors can be used to emphasize groupings. Look at the microwave control

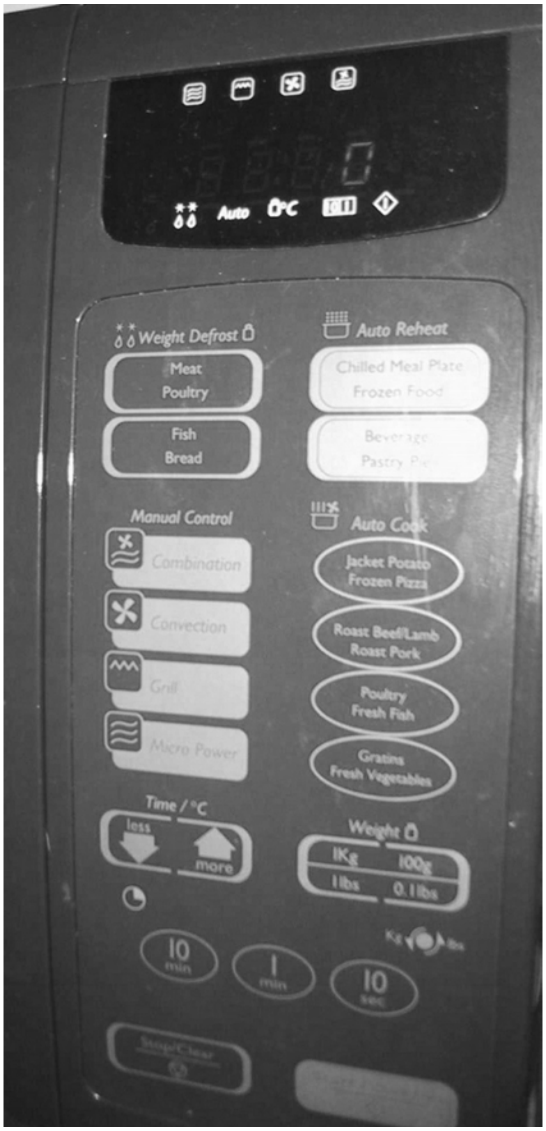


Figure 5.9 Microwave control panel

panel in Figure 5.9. See how the buttons differ in using the foreground and background colors (green and gold) so that groups are associated with one another. See also how the buttons are laid out to separate them into groups of similar function.

Alignment

Alignment of lists is also very important. For users who read text from left to right, lists of text items should normally be aligned to the left. Numbers, however, should

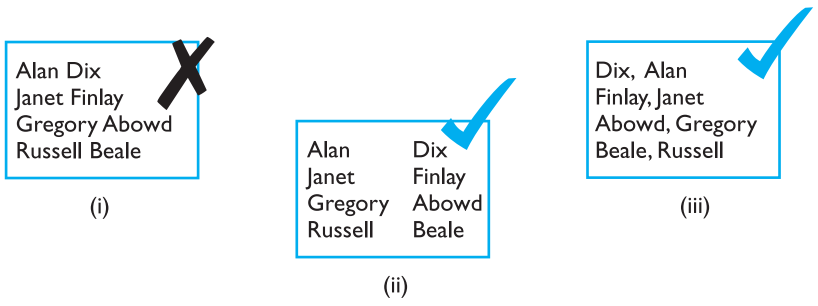


Figure 5.10 Looking up surnames

normally be aligned to the right (for integers) or at the decimal point. This is because the shape of the column then gives an indication of magnitude – a sort of mini-histogram. Items like names are particularly difficult. Consider list (i) in Figure 5.10. It is clearly hard to look someone up if you only know their surname. To make it easy, such lists should be laid out in columns as in (ii), or have forename and surname reversed as in (iii). (The dates in Figure 5.13, Section 5.7.3, pose similar problems, as the years do not align, even when the folder is sorted by date.)

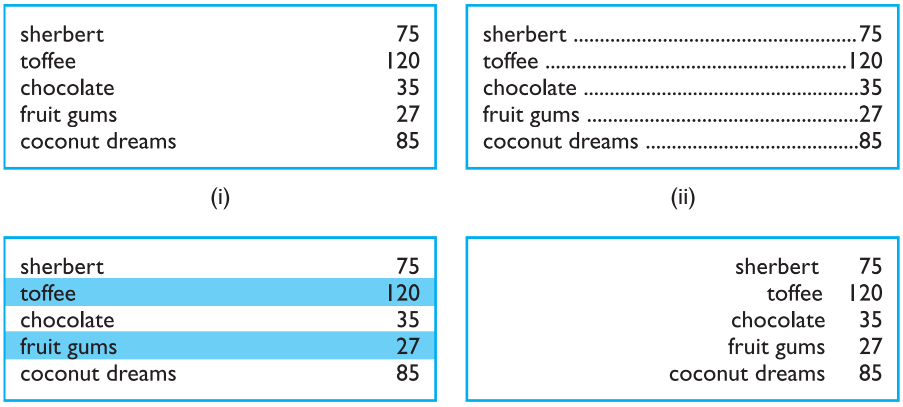
Design Focus

Alignment and layout matter

Look quickly at these two columns of numbers and try to find the biggest number in each column.

```
   532.56     627.865
    179.3       1.005763
  256.317     382.583
       15    2502.56
   73.948     432.935
     1035       2.0175
    3.142     652.87
 497.6256      56.34
```

Multiple column lists require more care. Text columns have to be wide enough for the largest item, which means you can get large gaps between columns. Figure 5.11 shows an example of this (i), and you can see how hard this makes it for your eye to scan across the rows. There are several visual ways to deal with this including: (ii) 'leaders' – lines of dots linking the columns; and (iii) using soft tone grays or colors behind rows or columns. This is also a time when it may be worth breaking other



(i)

sherbert	75
toffee	120
chocolate	35
fruit gums	27
coconut dreams	85

(ii)

sherbert	75
toffee	120
chocolate	35
fruit gums	27
coconut dreams	85

(iii)

sherbert	75
toffee	120
chocolate	35
fruit gums	27
coconut dreams	85

(iv)

sherbert	75
toffee	120
chocolate	35
fruit gums	27
coconut dreams	85

Figure 5.11 Managing multiple columns

alignment rules, perhaps right aligning some text items as in (iv). This last alternative might be a good solution if you were frequently scanning the numbers and only occasionally scanning the names of items, but not if you needed frequently to look up names (which anyway are not sorted in this figure!). You can also see that this is an example of a design trade-off – good alignment for individual columns versus ability to see relationship across rows.

White space

In typography the space between the letters is called the counter. In painting this is also important and artists may focus as much on the space between the foreground elements such as figures and buildings as on the elements themselves. Often the shape of the counter is the most important part of the composition of a painting and in calligraphy and typography the balance of a word is determined by giving an even weight to the counters. If one ignores the 'content' of a screen and instead concentrates on the counter – the space between the elements – one can get an overall feel for the layout. If elements that are supposed to be related look separate when you focus on the counter, then something is wrong. Screwing up your eyes so that the screen becomes slightly blurred is another good technique for taking your attention away from the content and looking instead at the broad structure.

Space can be used in several ways. Some of these are shown in Figure 5.12. The colored areas represent continuous areas of text or graphics. In (i) we can see space used to separate blocks as you often see in gaps between paragraphs or space between sections in a report. Space can also be used to create more complex structures. In (ii) there are clearly four main areas: ABC, D, E and F. Within one of these are three further areas, A, B and C, which themselves are grouped as A on its own, followed by B and C together. In Figure 5.12 (iii), we can see space used to highlight. This is a technique used frequently in magazines to highlight a quote or graphic.

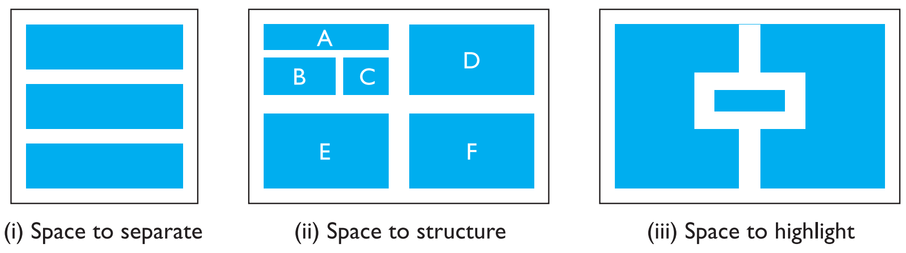


Figure 5.12 Using white space in layout

5.7.2 User action and control

Entering information

Some of the most complicated and difficult screen layouts are found in forms-based interfaces and dialog boxes. In each case the screen consists not only of information presented to the user, but also of places for the user to enter information or select options. Actually, many of the same layout issues for data presentation also apply to fields for data entry. Alignment is still important. It is especially common to see the text entry boxes aligned in a jagged fashion because the field names are of different lengths. This is an occasion where right-justified text for the field labels may be best or, alternatively, in a graphical interface a smaller font can be used for field labels and the labels placed just above and to the left of the field they refer to.

For both presenting and entering information a clear logical layout is important. The task analysis techniques in Chapter 15 can help in determining how to group screen items and also the order in which users are likely to want to read them or fill them in. Knowing also that users are likely to read from left to right and top to bottom (depending on their native language!) means that a screen can be designed so that users encounter items in an appropriate order for the task at hand.

Knowing what to do

Some elements of a screen are passive, simply giving you information; others are active, expecting you to fill them in, or do something to them. It is often not even clear which elements are active, let alone what the effect is likely to be when you interact with them!

This is one of the reasons for platform and company style guides. If everyone designs buttons to look the same and menus to look the same, then users will be able to recognize them when they see them. However, this is not sufficient in itself. It is important that the labels and icons on menus are also clear. Again, standards can help for common actions such as save, delete or print. For more system-specific actions, one needs to follow broader principles. For example, a button says 'bold': does this represent the current *state* of a system or the *action* that will be performed if the button is pressed?

Affordances

These are especially difficult problems in multimedia applications where one may deliberately adopt a non-standard and avant-garde style. How are users supposed to know where to click? The psychological idea of *affordance* says that things may suggest by their shape and other attributes what you can do to them: a handle affords pulling or lifting; a button affords pushing. These affordances can be used when designing novel interaction elements. One can either mimic real-world objects directly, or try to emulate the critical aspects of those objects. What you must not do is depict a real-world object in a context where its normal affordances do not work!

Note also that affordances are not intrinsic, but depend on the background and culture of users. Most computer-literate users will click on an icon. This is not because they go around pushing pictures in art galleries, but because they have learned that this is an affordance of such objects in a computer domain. Similarly, such experienced users may well double click if a single click has no effect, yet novices would not even think of double clicking – after all, double clicking on most real buttons turns them off again!

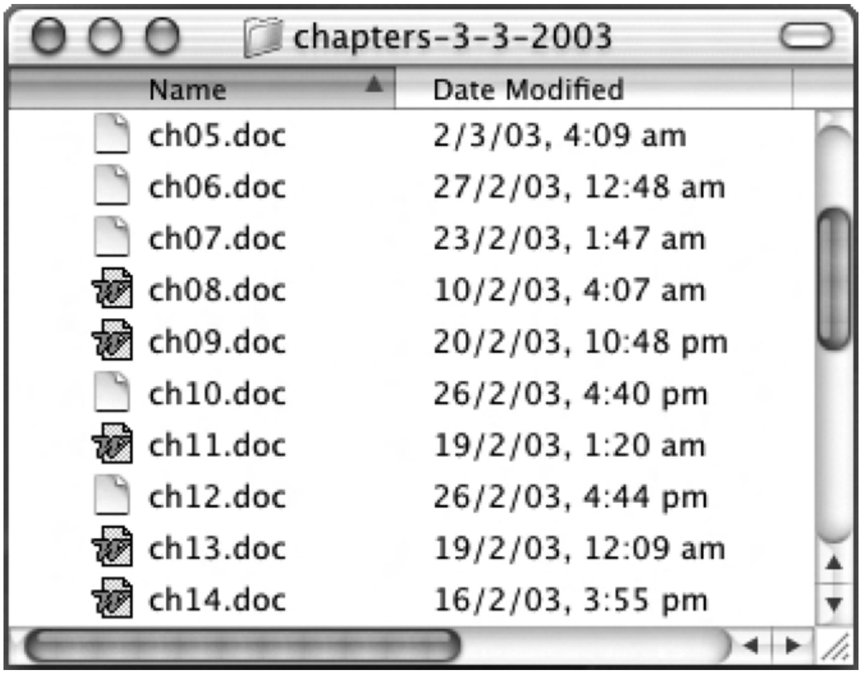
5.7.3 Appropriate appearance

Presenting information

The way of presenting information on screen depends on the kind of information: text, numbers, maps, tables; on the technology available to present it: character display, line drawing, graphics, virtual reality; and, most important of all, on the purpose for which it is being used. Consider the window in Figure 5.13. The file listing is alphabetic, which is fine if we want to look up the details of a particular file, but makes it very difficult to find recently updated files. Of course, if the list were ordered by date then it would be difficult to find a particular file. Different purposes require different representations. For more complex numerical data, we may be considering scatter graphs, histograms or 3D surfaces; for hierarchical structures, we may consider outlines or organization diagrams. But, no matter how complex the data, the principle of matching presentation to purpose remains.

The issue of presentation has been around for many years, long before computers, interactive systems or HCI! Probably the best source for this issue is Tufte's book [351]. It is targeted principally at static presentations of information, as in books, but most design principles transfer directly.

We have an advantage when presenting information in an interactive system in that it is easy to allow the user to choose among several representations, thus making it possible to achieve different goals. For example, with Macintosh folder windows (as in Figure 5.13) the user can click on a column heading and the file list is reordered, so one can look at the files by, say, name or date. This is not an excuse for ignoring the user's purpose, but means that we can plan for a range of possible uses.


chapters-3-3-2003

Name	Date Modified
ch05.doc	2/3/03, 4:09 am
ch06.doc	27/2/03, 12:48 am
ch07.doc	23/2/03, 1:47 am
ch08.doc	10/2/03, 4:07 am
ch09.doc	20/2/03, 10:48 pm
ch10.doc	26/2/03, 4:40 pm
ch11.doc	19/2/03, 1:20 am
ch12.doc	26/2/03, 4:44 pm
ch13.doc	19/2/03, 12:09 am
ch14.doc	16/2/03, 3:55 pm

Figure 5.13 Alphabetic file listing. Screen shot reprinted by permission from Apple Computer, Inc.

Aesthetics and utility

Remember that a pretty interface is not necessarily a good interface. Ideally, as with any well-designed item, an interface should be aesthetically pleasing. Indeed, good graphic design and attractive displays can increase users' satisfaction and thus improve productivity.

However, beauty and utility may sometimes be at odds. For example, an industrial control panel will often be built up of the individual controls of several subsystems, some designed by different teams, some bought in. The resulting inconsistency in appearance may look a mess and suggest tidying up. Certainly some of this inconsistency may cause problems. For example, there may be a mix of telephone-style and calculator-style numeric keypads. Under stress it would be easy to mis-key when swapping between these. However, the diversity of controls can also help the operator keep track of which controls refer to which subsystem – any redesign must preserve this advantage.

The conflict between aesthetics and utility can also be seen in many 'well-designed' posters and multimedia systems. In particular, the backdrop behind text must have low contrast in order to leave the text readable; this is often not the case and graphic designers may include excessively complex and strong backgrounds because they look good. The results are impressive, perhaps even award winning, but completely unusable!

On a more positive note, careful application of aesthetic concepts can also aid comprehensibility. An example of this is the idea of the counter and use of space that we discussed earlier. In consumer devices these aesthetic considerations may often be the key differentiator between products, for example, the sleek curves of a car. This is not missed by designers of electronic goods: devices are designed to be good to touch and feel as well as look at and this is certainly one of the drivers for the futuristic shapes of the Apple iMac family.

THE COUNTER

Making a mess of it: color and 3D

One of the worst features in many interfaces is their appalling use of color. This is partly because many monitors only support a limited range of primary colors and partly because, as with the overuse of different fonts in word processors, the designer got carried away. Aside from issues of good taste, an overuse of color can be distracting and, remembering from Chapter 1 that a significant proportion of the population is color blind, may mean that parts of the text are literally invisible to some users. In general, color should be used sparingly and not relied upon to give information, but rather to reinforce other attributes.

The increasing use of 3D effects in interfaces has posed a whole new set of problems for text and numerical information. Whilst excellent for presenting physical information and certain sorts of graphs, text presented in perspective can be very difficult to read and the all too common 3D pie chart is all but useless. We will discuss ways to make 3D actually useful for visualization in Chapter 20.

Design Focus

Checking screen colors

Even non-color-blind users will find it hard to read text where the intensity of the text and background are similar. A good trick is to adjust the color balance on your monitor so that it is reduced to grays, or to print screens on a black and white printer. If your screen is unreadable in grayscale then it is probably difficult to read in full color.

Localization / internationalization

If you are working in a different country, you might see a document being word processed where the text of the document and the file names are in the local language, but all the menus and instructions are still in English. The process of making software suitable for different languages and cultures is called *localization* or *internationalization*.

It is clear that words have to change and many interface construction toolkits make this easy by using *resources*. When the program uses names of menu items, error messages and other text, it does not use the text directly, but instead uses a resource identifier, usually simply a number. A simple database is constructed separately that binds these identifiers to particular words and phrases. A different resource database is constructed for each language, and so the program can be customized to use in a particular country by simply choosing the appropriate resource database.

However, changing the language is only the simplest part of internationalization. Much of the explicit guidance on alignment and layout is dependent on a left-to-right, top-to-bottom language such as English and most European languages. This obviously changes completely for other types of language. Furthermore, many icons and images are only meaningful within a restricted cultural context. Despite the apparent international hegemony of Anglo-American culture, one cannot simply assume that its symbols and norms will be universally understood. A good example of this is the use of ticks ✓ and crosses ✗. In Anglo-American culture these represent opposites, positive and negative, whereas in most of Europe the two are interchangeable.

5.8 ITERATION AND PROTOTYPING

Because human situations are complex and designers are not infallible it is likely that our first design will not be perfect! For this reason, almost all interaction design includes some form of iteration of ideas. This often starts early on with paper designs and storyboards being demonstrated to colleagues and potential users. Later in the design process one might use mockups of physical devices or tools such as Shockwave or Visual Basic to create prototype versions of software.

Any of these prototypes, whether paper-based or running software, can then be evaluated to see whether they are acceptable and where there is room for improvement. This sort of evaluation, intended to improve designs, is called *formative* evaluation. This is in contrast to *summative* evaluation, which is performed at the end to verify whether the product is good enough. Chapter 9 considers evaluation in detail. One approach is to get an expert to use a set of guidelines, for example the 'knowing where you are' list above, and look screen by screen to see if there are any violations. The other main approach is to involve real users either in a controlled experimental setting, or 'in the wild' – a real-use environment.

The result of evaluating the system will usually be a list of faults or problems and this is followed by a redesign exercise, which is then prototyped, evaluated . . . Figure 5.14 shows this process. The end point is when there are no more problems that can economically be fixed.

So iteration and prototyping are the universally accepted 'best practice' approach for interaction design. However, there are some major pitfalls of prototyping, rarely acknowledged in the literature.

Prototyping is an example of what is known as a *hill-climbing* approach. Imagine you are standing somewhere in the open countryside. You walk uphill and keep going uphill as steeply as possible. Eventually you will find yourself at a hill top. This

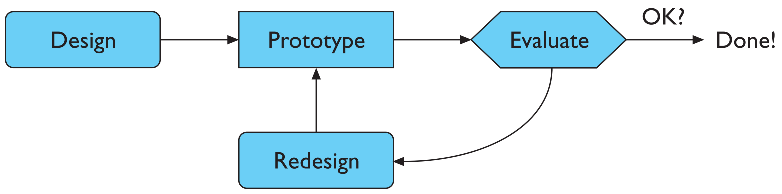


Figure 5.14 Role of prototyping

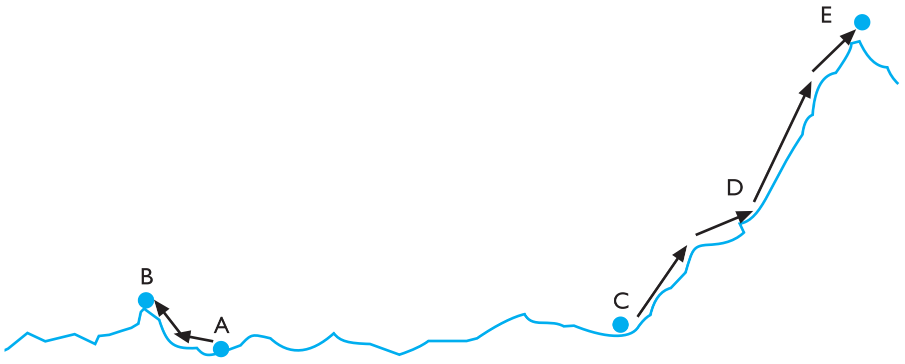


Figure 5.15 Moving little by little . . . but to where?

is exactly how iterative prototyping works: you start somewhere, evaluate it to see how to make it better, change it to make it better and then keep on doing this until it can't get any better.

However, hill climbing doesn't always work. Imagine you start somewhere near Cambridge, UK. If you keep moving uphill (and it is very difficult to work out which direction that is because it is very flat!), then eventually you would end up at the top of the Gog Magog hills, the nearest thing around . . . all of 300 feet. However, if you started somewhere else you might end up at the top of the Matterhorn. Hill-climbing methods always have the potential to leave you somewhere that is the best in the immediate area, but very poor compared with more distant places. Figure 5.15 shows this schematically: if you start at A you get trapped at the *local maximum* at B, but if you start at C you move up through D to the *global maximum* at E.

This problem of getting trapped at local maxima is also possible with interfaces. If you start with a bad design concept you may end at something that is simply a tidied up version of that bad idea!

From this we can see that there are two things you need in order for prototyping methods to work:

1. To understand what is wrong and how to improve.
2. A good start point.

The first is obvious; you cannot iterate the design unless you know what must be done to improve it. The second, however, is needed to avoid local maxima. If you wanted to climb as high as you could, you would probably book a plane to the Himalayas, not Cambridgeshire.

A really good designer might guess a good initial design based on experience and judgment. However, the complexity of interaction design problems means that this insight is hard. Another approach, very common in graphical design, is to have several initial design ideas and drop them one by one as they are developed further. This is a bit like parachuting 10 people at random points of the earth. One of them is perhaps likely to end up near a high mountain.

One of the things that theoretical methods and models, as found in Part 3 of this book, can do is to help us with both (1) and (2).

5.9 SUMMARY

We have seen that design in HCI is not just about creating devices or software, but instead is about the whole interaction between people, software and their environment. Because of this it is good to see the product of design not just as the obvious artifacts but as the whole intervention that changes the existing situation to a new one.

In Section 5.2, design was defined as 'achieving goals within constraints'. In the case of interaction design the goals are about improving some aspect of work, home or leisure using technology. The constraints remind us that the final design will inevitably involve trade-offs between different design issues and furthermore should never be 'perfect' as cost and timeliness should prevent indefinite tinkering. To achieve good design we must understand our materials and in the case of interaction design these materials include not just the computers and technical devices, but also humans. If we treated humans in design with only as much care as physical materials it is clear that 'human error' after accidents would be regarded as 'design error' – a good designer understands the natural limitations of ordinary people.

Section 5.3 gave a bird's-eye view of the design process, which gives a context for much of the rest of this book.

The process starts with understanding the situation as it is and the requirements for change. Section 5.4 provided some simple techniques for dealing with this: getting to know your users, who they are, remembering that they are different from you, but trying to imagine what it is like for them. You can talk to users, but you should also observe them in other ways, as we are all bad at articulating what we do. One way to help retain a user focus in design is to use personae – detailed word pictures of imaginary but typical users.

Section 5.5 introduced scenarios and rich stories about design, which can help us explore the design space and to discuss potential designs with other designers and potential users. Both scenarios and personae need to be vivid and to include rich contextual details – not just a record of user actions on the system!

The details of potential designs need to be worked out and in Section 5.6 we looked at the overall navigation design of the system. We started by looking at local structure, the way one screen, page or state of an application relates to those it immediately links to. The users need to know where they are, what they can do, what will happen when they do things, and what has happened in the past. This can aid users as they goal seek, or move closer towards their goals without having to necessarily understand completely the whole route there. The global structure of the application is also important. We saw how hierarchy diagrams can give a logical view of an application, which can be used to design menu or site structures. In contrast, the user dialog focusses on the flow of user and system actions. One way to do this is using network diagrams of screens or states of the system and how they link to one another. Any designed system must also relate to its environment: other applications, other websites, other physical devices.

In Section 5.7 we looked at screen design and layout. We saw that there were various visual tools that could help us to ensure that the physical structure of our screen emphasized the logical structure of the user interaction. These tools included physical grouping, ordering of items, decoration such as fonts, lines and color, alignment and the use of white space. These are important both for appropriate display of information and to lay out controls and data entry fields for ease of use. It is also important that controls have appropriate affordances – that is have visual and tactile attributes that suggest their use. Information presented on screen, whether individual items, tabular or graphical, should be appropriate to the user's purpose and this may mean allowing interactions to change the layout, for example re-sort tables by different columns. Aesthetics are also important, but may conflict with utility. Depending on the context you may need to make different trade-offs between these. Good graphical design is an area and a skill all of its own, but some features such as bad use of color and 3D effects are bad for both aesthetics and usability!

Finally, in Section 5.8, we saw that iteration is an essential part of virtually any interaction design process because we cannot get things right first time. However, iterative methods may get trapped in local maxima. To make iterative processes work, we need either extensive personal experience or theoretical understanding to help us get better initial designs.

Exercises

5.1 Use a pocket alarm clock or wristwatch to set yourself alarms every 15 minutes one working day. Write down exactly what you are doing. How surprising is it?

Exercises 5.2, 5.3, 5.4 and 5.5 are based around a nuclear reactor scenario on the book website at: /e3/scenario/nuclear/ You will need to read the scenario in order to answer these exercises.

5.2 Comment on the user of color in the Alarm Control, Emergency Shutdown and Emergency Confirm panels (Figure CS.2 – for figures, see the web scenario).

5.3 Comment on the use of layout and other elements in the control panels (Figures CS.1, CS.2 and CS.3), including the way in which various visual elements support or hinder logical grouping and sequence.

5.4 Working through the accident scenario, explain why the various problems arise.

5.5 Suggest potential ways of improving the interface to avoid a similar problem recurring.

RECOMMENDED READING

J. Preece, Y. Rogers and H. Sharp, *Interaction Design: Beyond Human–Computer Interaction*, John Wiley, 2002.
A general textbook on interaction design with especially strong focus on evaluation.

J. Carroll, editor, *Interacting with Computers*, Vol. 13, No. 1, special issue on 'Scenario-based system development', 2000.
Contributions from several authors on using scenarios in design.

J. Carroll, *Making Use: Scenario-Based Design of Human–Computer Interactions*, MIT Press, 2000.
John Carroll's own book dedicated solely to using scenarios in design.

J. McGrenere and W. Ho, Affordances: clarifying and evolving a concept, *Proceedings of Graphics Interface 2000*, pp. 179–86, 2000.
This paper reviews all the major work on affordances from Gibson's original definition and focusses especially on Norman's popularization of the word which has been the way many encounter it. It also reviews the work of Bill Gaver, who is probably the first person to use affordance as a concept within HCI.

E. Tufte, *Envisioning Information*, Graphics Press, Cheshire, USA, 1990, and
E. Tufte, *The Visual Display of Quantitative Information*, Graphics Press, Cheshire, USA, 1997.
Tufte's books are the 'must read' for graphical presentation, packed with examples and pictures – from timetables to Napoleon's disastrous Russian campaign.

HCI IN THE SOFTWARE PROCESS

6

OVERVIEW

- Software engineering provides a means of understanding the structure of the design process, and that process can be assessed for its effectiveness in interactive system design.
- Usability engineering promotes the use of explicit criteria to judge the success of a product in terms of its usability.
- Iterative design practices work to incorporate crucial customer feedback early in the design process to inform critical decisions which affect usability.
- Design involves making many decisions among numerous alternatives. Design rationale provides an explicit means of recording those design decisions and the context in which the decisions were made.

6.1 INTRODUCTION

In Chapter 4 we concentrated on identifying aspects of usable interactive systems by means of concrete examples of successful paradigms. The design goal is to provide reliable techniques for the repeated design of successful and usable interactive systems. It is therefore necessary that we go beyond the exercise of identifying paradigms and examine the process of interactive system design. In the previous chapter we introduced some of the elements of a user-centered design process. Here we expand on that process, placing the design of interactive systems within the established frameworks of software development.

Within computer science there is already a large subdiscipline that addresses the management and technical issues of the development of software systems – called *software engineering*. One of the cornerstones of software engineering is the *software life cycle*, which describes the activities that take place from the initial concept formation for a software system up until its eventual phasing out and replacement. This is not intended to be a software engineering textbook, so it is not our major concern here to discuss in depth all of the issues associated with software engineering and the myriad life-cycle models.

The important point that we would like to draw out is that issues from HCI affecting the usability of interactive systems are relevant within all the activities of the software life cycle. Therefore, software engineering for interactive system design is not simply a matter of adding one more activity that slots in nicely with the existing activities in the life cycle. Rather, it involves techniques that span the entire life cycle.

We will begin this chapter by providing an introduction to some of the important concepts of software engineering, in Section 6.2. Specifically, we will describe the major activities within the traditional software life cycle and discuss the issues raised by the special needs of interactive systems. We will then describe some specific approaches to interactive system design, which are used to promote product usability throughout the life cycle. In Section 6.3, we will discuss a particular methodology called *usability engineering* in which explicit usability requirements are used as goals for the design process. In Section 6.4, we consider iterative design practices that involve prototyping and participative evaluation. We conclude this chapter with a discussion of *design rationale*. Design is a decision-making activity and it is important to keep track of the decisions that have been made and the context in which they were made. Various design rationale techniques, presented in Section 6.5, are used to support this critical activity.

6.2 THE SOFTWARE LIFE CYCLE

One of the claims for software development is that it should be considered as an engineering discipline, in a way similar to how electrical engineering is considered for hardware development. One of the distinguishing characteristics of any engineering discipline is that it entails the structured application of scientific techniques to the development of some product. A fundamental feature of software engineering, therefore, is that it provides the structure for applying techniques to develop software systems. The software life cycle is an attempt to identify the activities that occur in software development. These activities must then be ordered in time in any development project and appropriate techniques must be adopted to carry them through.

In the development of a software product, we consider two main parties: the customer who requires the use of the product and the designer who must provide the product. Typically, the customer and the designer are groups of people and some people can be both customer and designer. It is often important to distinguish between the customer who is the client of the designing company and the customer who is the eventual user of the system. These two roles of customer can be played by different people. The group of people who negotiate the features of the intended system with the designer may never be actual users of the system. This is often particularly true of web applications. In this chapter, we will use the term 'customer' to refer to the group of people who interact with the design team and we will refer to those who will interact with the designed system as the user or end-user.

6.2.1 Activities in the life cycle

A more detailed description of the life cycle activities is depicted in Figure 6.1. The graphical representation is reminiscent of a waterfall, in which each activity naturally leads into the next. The analogy of the waterfall is not completely faithful to the real relationship between these activities, but it provides a good starting point for discussing the logical flow of activity. We describe the activities of this waterfall model of the software life cycle next.[1]

Requirements specification

In requirements specification, the designer and customer try to capture a description of *what* the eventual system will be expected to provide. This is in contrast to determining *how* the system will provide the expected services, which is the concern of later activities. Requirements specification involves eliciting information from the customer about the work environment, or domain, in which the final product will function. Aspects of the work domain include not only the particular functions that the software product must perform but also details about the environment in which it must operate, such as the people whom it will potentially affect and the new product's relationship to any other products which it is updating or replacing.

Requirements specification begins at the start of product development. Though the requirements are from the customer's perspective, if they are to be met by the

1 Some authors distinguish between the software development process and the software life cycle, the waterfall model being used to describe the former and not the latter. The main distinction for our purposes is that operation and maintenance of the product is not part of the development process.

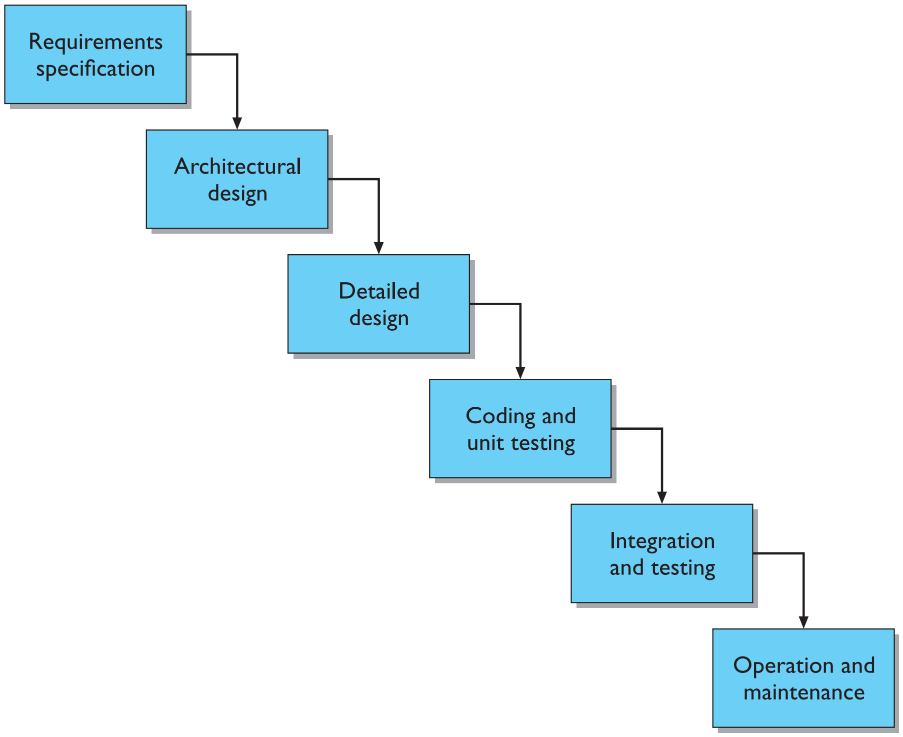


Figure 6.1 The activities in the waterfall model of the software life cycle

software product they must be formulated in a language suitable for implementation. Requirements are usually initially expressed in the native language of the customer. The executable languages for software are less natural and are more closely related to a mathematical language in which each term in the language has a precise interpretation, or semantics. The transformation from the expressive but relatively ambiguous natural language of requirements to the more precise but less expressive executable languages is one key to successful development. In Chapter 15 we discuss task analysis techniques, which are used to express work domain requirements in a form that is both expressive and precise.

Architectural design

As we mentioned, the requirements specification concentrates on what the system is supposed to do. The next activities concentrate on *how* the system provides the services expected from it. The first activity is a high-level decomposition of the system into components that can either be brought in from existing software products or be developed from scratch independently. An architectural design performs this decomposition. It is not only concerned with the functional decomposition of the system, determining which components provide which services. It must also describe the interdependencies between separate components and the sharing of resources that will arise between components.

There are many structured techniques that are used to assist a designer in deriving an architectural description from information in the requirements specification (such as CORE, MASCOT and HOOD). Details of these techniques are outside the scope of this book, but can be found in any good software engineering textbook. What we will mention here is that the majority of these techniques are adequate for capturing the *functional requirements* of the system – the services the system must provide in the work domain – but do not provide an immediate way to capture other *non-functional requirements* – features of the system that are not directly related to the actual services provided but relate to the manner in which those services must be provided. Some classic examples of non-functional requirements are the efficiency, reliability, timing and safety features of the system. Interactive features of the system, such as those that will be described by the principles in Chapter 7, also form a large class of non-functional requirements.

Detailed design

The architectural design provides a decomposition of the system description that allows for isolated development of separate components which will later be integrated. For those components that are not already available for immediate integration, the designer must provide a sufficiently detailed description so that they may be implemented in some programming language. The detailed design is a *refinement* of the component description provided by the architectural design. The behavior implied by the higher-level description must be preserved in the more detailed description.

Typically, there will be more than one possible refinement of the architectural component that will satisfy the behavioral constraints. Choosing the best refinement is often a matter of trying to satisfy as many of the non-functional requirements of the system as possible. Thus the language used for the detailed design must allow some analysis of the design in order to assess its properties. It is also important to keep track of the design options considered, the eventual decisions that were made and the reasons why, as we will discuss in Section 6.5 on design rationale.

Coding and unit testing

The detailed design for a component of the system should be in such a form that it is possible to implement it in some executable programming language. After coding, the component can be tested to verify that it performs correctly, according to some test criteria that were determined in earlier activities. Research on this activity within the life cycle has concentrated on two areas. There is plenty of research that is geared towards the automation of this coding activity directly from a low-level detailed design. Most of the work in *formal methods* operates under the hypothesis that, in theory, the transformation from the detailed design to the implementation is from one mathematical representation to another and so should be able to be entirely automated. Other, more practical work concentrates on the automatic generation of tests from output of earlier activities which can be performed on a piece of code to verify that it behaves correctly.

Integration and testing

Once enough components have been implemented and individually tested, they must be integrated as described in the architectural design. Further testing is done to ensure correct behavior and acceptable use of any shared resources. It is also possible at this time to perform some acceptance testing with the customers to ensure that the system meets their requirements. It is only after acceptance of the integrated system that the product is finally released to the customer.

It may also be necessary to certify the final system according to requirements imposed by some outside authority, such as an aircraft certification board. As of 1993, a European health and safety act requires that all employers provide their staff with usable systems. The international standards authority, ISO, has also produced a standard (ISO 9241) to define the usability of office environment workstations. Coupled together, the health and safety regulations and ISO 9241 provide impetus for designers to take seriously the HCI implications of their design.

Maintenance

After product release, all work on the system is considered under the category of maintenance, until such time as a new version of the product demands a total redesign or the product is phased out entirely. Consequently, the majority of the lifetime of a product is spent in the maintenance activity. Maintenance involves the correction of errors in the system which are discovered after release and the revision of the system services to satisfy requirements that were not realized during previous development. Therefore, maintenance provides feedback to all of the other activities in the life cycle, as shown in Figure 6.2.

6.2.2 Validation and verification

Throughout the life cycle, the design must be checked to ensure that it both satisfies the high-level requirements agreed with the customer and is also complete and internally consistent. These checks are referred to as *validation* and *verification*, respectively. Boehm [36a] provides a useful distinction between the two, characterizing validation as designing 'the right thing' and verification as designing 'the thing right'. Various languages are used throughout design, ranging from informal natural language to very precise and formal mathematical languages. Validation and verification exercises are difficult enough when carried out within one language; they become much more difficult, if not impossible, when attempted between languages.

Verification of a design will most often occur within a single life-cycle activity or between two adjacent activities. For example, in the detailed design of a component

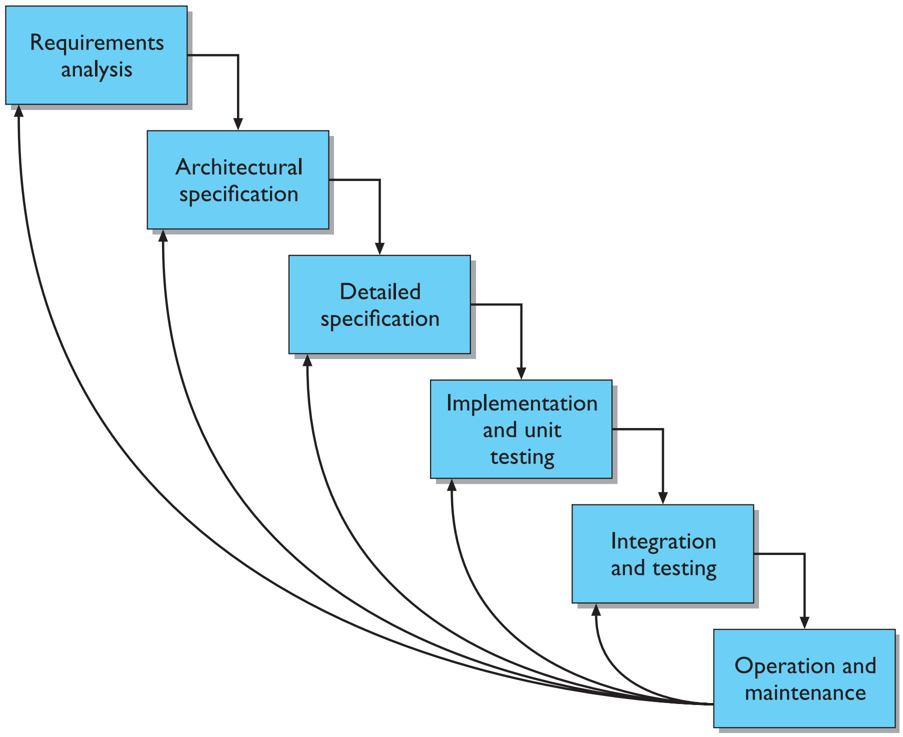

Requirements analysis
Architectural specification
Detailed specification
Implementation and unit testing
Integration and testing
Operation and maintenance

Figure 6.2 Feedback from maintenance activity to other design activities

of a payroll accounting system, the designer will be concerned with the correctness of the algorithm to compute taxes deducted from an employee's gross income. The architectural design will have provided a general specification of the information input to this component and the information it should output. The detailed description will introduce more information in refining the general specification. The detailed design may also have to change the representations for the information and will almost certainly break up a single high-level operation into several low-level operations that can eventually be implemented. In introducing these changes to information and operations, the designer must show that the refined description is a legal one within its language (internal consistency) and that it describes all of the *specified* behavior of the high-level description (completeness) in a provably correct way (relative consistency).

Validation of a design demonstrates that within the various activities the customer's requirements are satisfied. Validation is a much more subjective exercise than verification, mainly because the disparity between the language of the requirements and the language of the design forbids any objective form of proof. In interactive system design, the validation against HCI requirements is often referred to as evaluation and can be performed by the designer in isolation or in cooperation with the customer. We discuss evaluation in depth in Chapter 9.

An important question, which applies to both verification and validation, asks exactly what constitutes a proof. We have repeatedly mentioned the language used in any design activity and the basis for the semantics of that language. Languages with a mathematical foundation allow reasoning and proof in the objective sense. An argument based entirely within some mathematical language can be accepted or refuted based upon universally accepted measures. A proof can be entirely justified by the rules of the mathematical language, in which case it is considered a formal proof. More common is a rigorous proof, which is represented within some mathematical language but which relies on the understanding of the reader to accept its correctness without appeal to the full details of the argument, which could be provided but usually are not. The difference between formality and rigour is in the amount of detail the prover leaves out while still maintaining acceptance of the proof.

Proofs that are for verification of a design can frequently occur within one language or between two languages which both have a precise mathematical semantics. Time constraints for a design project and the perceived economic implications of the separate components usually dictate which proofs are carried out in full formality and which are done only rigorously (if at all). As research in this area matures and automated tools provide assistance for the mechanical aspects of proof, the cost of proof should decrease.

Validation proofs are much trickier, as they almost always involve a transformation between languages. Furthermore, the origin of customer requirements arises in the inherent ambiguity of the real world and not the mathematical world. This precludes the possibility of objective proof, rigorous or formal. Instead, there will always be a leap from the informal situations of the real world to any formal and structured development process. We refer to this inevitable disparity as the *formality gap*, depicted in Figure 6.3.

The formality gap means that validation will always rely to some extent on subjective means of proof. We can increase our confidence in the subjective proof by effective use of real-world experts in performing certain validation chores. These experts will not necessarily have design expertise, so they may not understand the

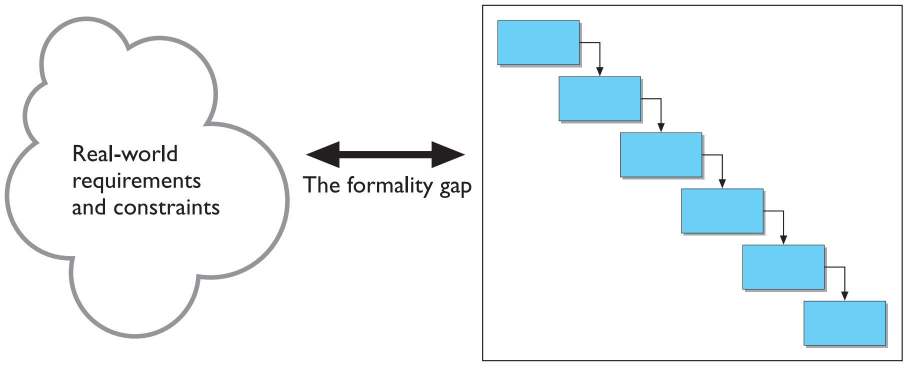


Figure 6.3 The formality gap between the real world and structured design

design notations used. Therefore, it is important that the design notations narrow the formality gap, making clear the claims that the expert can then validate. For interactive systems, the expert will have knowledge from a cognitive or psychological domain, so the design specification must be readily interpretable from a psychological perspective in order to validate it against interactive requirements of the system. We will discuss design techniques and notations that narrow the formality gap for validation of interactive properties of systems in Part 3.

6.2.3 Management and contractual issues

The life cycle described above concentrated on the more technical features of software development. In a technical discussion, managerial issues of design, such as time constraints and economic forces, are not as important. The different activities of the life cycle are logically related to each other. We can see that requirements for a system precede the high-level architectural design which precedes the detailed design, and so on. In reality, it is quite possible that some detailed design is attempted before all of the architectural design. In management, a much wider perspective must be adopted which takes into account the marketability of a system, its training needs, the availability of skilled personnel or possible subcontractors, and other topics outside the activities for the development of the isolated system.

As an example, we will take the development of a new aircraft on which there will be many software subsystems. The aircraft company will usually go through a concept evaluation period of up to 10 years before making any decision about actual product development. Once it has been decided to build a certain type of aircraft, loosely specified in the case of commercial aircraft in terms of passenger capacity and flight range, more explicit design activity follows. This includes joint analysis for both the specification of the aircraft and determination of training needs. It is only after the architectural specification of the aircraft is complete that the separate systems to be developed are identified. Some of these systems will be software systems, such as the flight management system or the training simulator, and these will be designed according to the life cycle described earlier. Typically, this will take four to five years. The separate aircraft systems are then integrated for ground and flight testing and certification before the aircraft is delivered to any customer airlines. The operating lifetime of an aircraft model is expected to be in the range of 20–40 years, during which time maintenance must be provided. The total lifetime of an aircraft from conception to phasing out is up to 55 years, only 4–5 years (excluding maintenance) of which contain the software life cycle which we are discussing in this chapter.

In managing the development process, the temporal relationship between the various activities is more important, as are the intermediate deliverables which represent the technical content, as the designer must demonstrate to the customer that progress is being made. A useful distinction, taken from McDermid [232], is that the technical perspective of the life cycle is described in *stages* of activity, whereas the managerial perspective is described in temporally bound *phases*. A phase is usually defined in terms of the documentation taken as input to the phase and the documentation delivered as output from the phase. So the requirements phase will take any marketing or conceptual development information, identifying potential customers, as input and produce a requirements specification that must be agreed upon between customer and designer.

This brings up another important issue from the management perspective. As the design activity proceeds, the customer and the designer must sign off on various documents, indicating their satisfaction with progress to date. These signed documents can carry a varying degree of contractual obligation between customer and designer. A signed requirements specification indicates both that the customer agrees to limit demands of the eventual product to those listed in the specification and also that the designer agrees to meet all of the requirements listed. From a technical perspective, it is easy to acknowledge that it is difficult, if not impossible, to determine all of the requirements before embarking on any other design activity. A satisfactory requirements specification may not be known until after the product has been in operation! From a management perspective, it is unacceptable to both designer and customer to delay the requirements specification that long.

So contractual obligation is a necessary consequence of managing software development, but it has negative implications on the design process as well. It is very difficult in the design of an interactive system to determine a priori what requirements to impose on the system to maximize its usability. Having to fix on some requirements too early will result either in general requirements that are very little guide for the designer or in specific requirements that compromise the flexibility of design without guaranteeing any benefits.

6.2.4 Interactive systems and the software life cycle

The traditional software engineering life cycles arose out of a need in the 1960s and 1970s to provide structure to the development of large software systems. In those days, the majority of large systems produced were concerned with data-processing applications in business. These systems were not highly interactive; rather, they were batch-processing systems. Consequently, issues concerning usability from an end-user's perspective were not all that important. With the advent of personal computing in the late 1970s and its huge commercial success and acceptance, most modern systems developed today are much more interactive, and it is vital to the success of any product that it be easy to operate for someone who is not expected to know much about how the system was designed. The modern user has a great amount of skill in the work that he performs without necessarily having that much skill in software development.

The life cycle for development we described above presents the process of design in a somewhat pipeline order. In reality, even for batch-processing systems, the actual design process is *iterative*, work in one design activity affecting work in any other activity both before or after it in the life cycle. We can represent this iterative relationship as in Figure 6.4, but that does not greatly enhance any understanding of the design process for interactive systems. You may ask whether it is worth the

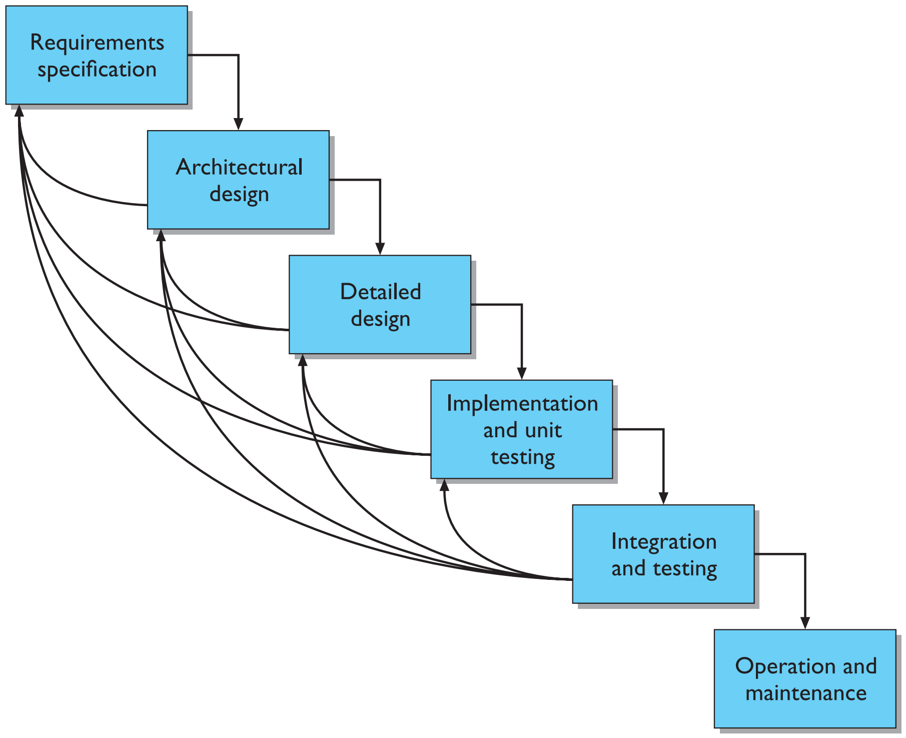

Requirements specification
Architectural design
Detailed design
Implementation and unit testing
Integration and testing
Operation and maintenance

Figure 6.4 Representing iteration in the waterfall model

intellectual effort to understand the interactive system design process. Is there really much design effort spent on the interactive aspects of a system to warrant our attention? A classic survey in 1978 by Sutton and Sprague at IBM resulted in an estimate that 50% of the designer's time was spent on designing code for the user interface [338]. A more recent and convincing survey by Myers and Rosson has confirmed that that finding holds true for the 1990s [247]. So it is definitely worth the effort to provide structure and techniques to understand, structure and improve the interactive design process! In this section, we will address features of interactive system design which are not treated properly by the traditional software life cycle.

The traditional software life cycle suits a principled approach to design; that is, if we know what it is we want to produce from the beginning, then we can structure our approach to design in order to attain the goal. We have already mentioned how, in practice, designers do not find out all of the requirements for a system before they begin. Figure 6.4 depicts how discovery in later activities can be reflected in iterations back to earlier stages. This is an admission that the requirements capture activity is not executed properly. The more serious claim we are making here is that all of the requirements for an interactive system *cannot* be determined from the start, and there are many convincing arguments to support this position. The result is that systems must be built and the interaction with users observed and evaluated in order to determine how to make them more usable.

Our models of the psychology and sociology of the human and human cognition, whether in isolation or in a group, are incomplete and do not allow us to predict how to design for maximum usability. There is much research on models of human users that allow prediction of their performance with interactive systems, which we will discuss in Chapter 12. These models, however, either rely on too much detail of the system to be useful at very early and abstract stages of design (see the section in Chapter 12 on the keystroke-level model) or they only apply to goal-oriented planned activity and not highly interactive WIMP systems (refer to the discussion at the end of Chapter 12).

This dearth of predictive psychological theory means that in order to test certain usability properties of their designs, designers must observe how actual users interact with the developed product and measure their performance. In order for the results of those observations to be worthwhile, the experiments must be as close to a real interaction situation as possible. That means the experimental system must be very much like it would be in the final product whose requirements the designer is trying to establish! As John Carroll has pointed out, the very detail of the actual system can crucially affect its usability, so it is not worthwhile to experiment on crude estimates of it, as that will provide observations whose conclusions will not necessarily apply to the real system [59].

One principled approach to interactive system design, which will be important in later chapters, relies on a clear understanding early on in the design of the tasks that the user wishes to perform. One problem with this assumption is that the tasks a user will perform are often only known by the user after he is familiar with the system on which he performs them. The chicken-and-egg puzzle applies to tasks and the artifacts on which he performs those tasks. For example, before the advent of word processors, an author would not have considered the use of a contracting and expanding outlining facility to experiment easily and quickly with the structure of a paper while it was being typed. A typewriter simply did not provide the ability to perform such a task, so how would a designer know to support such a task in designing the first word processor?

Also, some of the tasks a user performs with a system were never explicitly intended as tasks by its designer. Take the example of a graphics drawing package that separates the constructed picture into separate layers. One layer is used to build graphical pictures which are entire objects – a circle or a square, for instance – and can be manipulated as those objects and retain their object identity. The other layer is used to paint pictures which are just a collection of pixels. The user can switch between the layers in order to create very complex pictures which are part object, part painted scene. But because of the complex interplay between overlapping images between the two layers, it is also possible to hide certain parts of the picture when in one layer and reveal them in the other layer. Such a facility will allow the user to do simple simulations, such as showing the effect of shadowing when switching a light on and off. It is very doubtful that the designers were thinking explicitly of supporting such simulation or animation tasks when they were designing these graphics systems, which were meant to build complex, but static, pictures.

A final point about the traditional software life cycle is that it does not promote the use of notations and techniques that support the user's perspective of the interactive system. We discussed earlier the purpose of validation and the formality gap. It is very difficult for an expert on human cognition to predict the cognitive demands that an abstract design would require of the intended user if the notation for the design does not reflect the kind of information the user must recall in order to interact. The same holds for assessing the timing behavior of an abstract design that does not explicitly mention the timing characteristics of the operations to be invoked or their relative ordering. Though no structured development process will entirely eliminate the formality gap, the particular notations used can go a long way towards making validation of non-functional requirements feasible with expert assistance.

In the remaining sections of this chapter, we will describe various approaches to augment the design process to suit better the design of interactive systems. These approaches are categorized under the banner of *user-centered design*.

6.3 USABILITY ENGINEERING

One approach to user-centered design has been the introduction of explicit *usability engineering* goals into the design process, as suggested by Whiteside and colleagues at IBM and Digital Equipment Corporation [377] and by Nielsen at Bellcore [260, 261]. Engineering depends on interpretation against a shared background of meaning, agreed goals and an understanding of how satisfactory completion will be judged. The emphasis for usability engineering is in knowing exactly what criteria will be used to judge a product for its usability.

The ultimate test of a product's usability is based on measurements of users' experience with it. Therefore, since a user's direct experience with an interactive system is at the physical interface, focus on the actual user interface is understandable. The danger with this limited focus is that much of the work that is accomplished in interaction involves more than just the surface features of the systems used to perform that work. In reality, the whole functional architecture of the system and the cognitive capacity of the users should be observed in order to arrive at meaningful measures. But it is not at all simple to derive measurements of activity beyond the physical actions in the world, and so usability engineering is limited in its application.

In relation to the software life cycle, one of the important features of usability engineering is the inclusion of a usability specification, forming part of the requirements specification, that concentrates on features of the user–system interaction which contribute to the usability of the product. Various attributes of the system are suggested as gauges for testing the usability. For each attribute, six items are defined to form the usability specification of that attribute. Table 6.1 provides an example of a usability specification for the design of a control panel for a video cassette recorder (VCR), based on the technique presented by Whiteside, Bennett and Holtzblatt [377].

Table 6.1 Sample usability specification for undo with a VCR

Attribute:	Backward recoverability
Measuring concept:	Undo an erroneous programming sequence
Measuring method:	Number of explicit user actions to undo current program
Now level:	No current product allows such an undo
Worst case:	As many actions as it takes to program in mistake
Planned level:	A maximum of two explicit user actions
Best case:	One explicit cancel action

In this example, we choose the principle of recoverability, described fully in Chapter 7, as the particular usability attribute of interest. Recoverability refers to the ability to reach a desired goal after recognition of some error in previous interaction. The recovery procedure can be in either a backward or forward sense. Current VCR design has resulted in interactive systems that are notoriously difficult to use; the redesign of a VCR provides a good case study for usability engineering. In designing a new VCR control panel, the designer wants to take into account how a user might recover from a mistake he discovers while trying to program the VCR to record some television program in his absence. One approach that the designer decides to follow is to allow the user the ability to undo the programming sequence, reverting the state of the VCR to what it was before the programming task began.

The backward recoverability attribute is defined in terms of a *measuring concept*, which makes the abstract attribute more concrete by describing it in terms of the actual product. So in this case, we realize backward recoverability as the ability to undo an erroneous programming sequence. The *measuring method* states how the attribute will be measured, in this case by the number of explicit user actions required to perform the undo, regardless of where the user is in the programming sequence.

The remaining four entries in the usability specification then provide the agreed criteria for judging the success of the product based on the measuring method. The *now level* indicates the value for the measurement with the existing system, whether it is computer based or not. The *worst case* value is the lowest acceptable measurement for the task, providing a clear distinction between what will be acceptable and what will be unacceptable in the final product. The *planned level* is the target for the design and the *best case* is the level which is agreed to be the best possible measurement given the current state of development tools and technology.

In the example, the designers can look at their previous VCR products and those of their competitors to determine a suitable now level. In this case, it is determined that no current model allows an undo which returns the state of the VCR to what it was before the programming task. For example, if a VCR allows you three separate recording programs, once you begin entering a new program in the number 1 program slot, the VCR forgets the previous contents of that slot and so you cannot recover it unless you remember what it was and then reprogram it.

Table 6.2 Criteria by which measuring method can be determined (adapted from Whiteside, Bennett and Holtzblatt [377], Copyright 1988, reprinted with permission from Elsevier)

1.	Time to complete a task
2.	Per cent of task completed
3.	Per cent of task completed per unit time
4.	Ratio of successes to failures
5.	Time spent in errors
6.	Per cent or number of errors
7.	Per cent or number of competitors better than it
8.	Number of commands used
9.	Frequency of help and documentation use
10.	Per cent of favorable/unfavorable user comments
11.	Number of repetitions of failed commands
12.	Number of runs of successes and of failures
13.	Number of times interface misleads the user
14.	Number of good and bad features recalled by users
15.	Number of available commands not invoked
16.	Number of regressive behaviors
17.	Number of users preferring your system
18.	Number of times users need to work around a problem
19.	Number of times the user is disrupted from a work task
20.	Number of times user loses control of the system
21.	Number of times user expresses frustration or satisfaction

Determining the worst case value depends on a number of things. Usually, it should be no lower than the now level. The new product should provide some improvement on the current state of affairs, and so it seems that at least some of the usability attributes should provide worst case values that are better than the now level. Otherwise, why would the customer bother with the new system (unless it can be shown to provide the same usability at a fraction of the cost)? The designers in the example have determined that the minimal acceptable undo facility would require the user to perform as many actions as he had done to program in the mistake. This is a clear improvement over the now level, since it at least provides for the possibility of undo. One way to provide such a capability would be by including an undo button on the control panel, which would effectively reverse the previous non-undo action. The designers figure that they should allow for the user to do a complete restoration of the VCR state in a maximum of two explicit user actions, though they recognize that the best case, at least in terms of the number of explicit actions, would require only one.

Tables 6.2 and 6.3, adapted from Whiteside, Bennett and Holtzblatt [377], provide a list of measurement criteria which can be used to determine the measuring method for a usability attribute and the possible ways to set the worst/best case and planned/now level targets. Measurements such as those promoted by usability engineering are also called *usability metrics*.

Table 6.3 Possible ways to set measurement levels in a usability specification (adapted from Whiteside, Bennett and Holtzblatt [377], Copyright 1988, reprinted with permission from Elsevier)

Set levels with respect to information on:
1. an existing system or previous version
2. competitive systems
3. carrying out the task without use of a computer system
4. an absolute scale
5. your own prototype
6. user's own earlier performance
7. each component of a system separately
8. a successive split of the difference between best and worst values observed in user tests

Table 6.4 Examples of usability metrics from ISO 9241

Usability objective	Effectiveness measures	Efficiency measures	Satisfaction measures
Suitability for the task	Percentage of goals achieved	Time to complete a task	Rating scale for satisfaction
Appropriate for trained users	Number of power features used	Relative efficiency compared with an expert user	Rating scale for satisfaction with power features
Learnability	Percentage of functions learned	Time to learn criterion	Rating scale for ease of learning
Error tolerance	Percentage of errors corrected successfully	Time spent on correcting errors	Rating scale for error handling

The ISO standard 9241, described earlier, also recommends the use of usability specifications as a means of requirements specification. Table 6.4 gives examples of usability metrics categorized by their contribution towards the three categories of usability: effectiveness, efficiency and satisfaction.

6.3.1 Problems with usability engineering

The major feature of usability engineering is the assertion of explicit usability metrics early on in the design process which can be used to judge a system once it is delivered. There is a very solid argument which points out that it is only through empirical approaches such as the use of usability metrics that we can reliably build more usable systems. Although the ultimate yardstick for determining usability may be by observing and measuring user performance, that does not mean that these measurements are the best way to produce a predictive design process for usability.

The problem with usability metrics is that they rely on measurements of very specific user actions in very specific situations. When the designer knows what the actions and situation will be, then she can set goals for measured observations. However, at early stages of design, designers do not have this information. Take our example usability specification for the VCR. In setting the acceptable and unacceptable levels for backward recovery, there is an assumption that a button will be available to invoke the undo. In fact, the designer was already making an implicit assumption that the user would be making errors in the programming of the VCR. Why not address the origin of the programming errors, then maybe undo would not be necessary?

We should recognize another inherent limitation for usability engineering, that is it provides a means of satisfying usability specifications and not necessarily usability. The designer is still forced to understand why a particular usability metric enhances usability for real people. Again, in the VCR example, the designer assumed that fewer explicit actions make the undo operation easier. Is that kind of assumption warranted?

6.4 ITERATIVE DESIGN AND PROTOTYPING

A point we raised earlier is that requirements for an interactive system cannot be completely specified from the beginning of the life cycle. The only way to be sure about some features of the potential design is to build them and test them out on real users. The design can then be modified to correct any false assumptions that were revealed in the testing. This is the essence of *iterative design*, a purposeful design process which tries to overcome the inherent problems of incomplete requirements specification by cycling through several designs, incrementally improving upon the final product with each pass.

The problems with the design process, which lead to an iterative design philosophy, are not unique to the usability features of the intended system. The problem holds for requirements specification in general, and so it is a general software engineering problem, together with technical and managerial issues.

On the technical side, iterative design is described by the use of *prototypes*, artifacts that simulate or animate some but not all features of the intended system. There are three main approaches to prototyping:

Throw-away The prototype is built and tested. The design knowledge gained from this exercise is used to build the final product, but the actual prototype is discarded. Figure 6.5 depicts the procedure in using throw-away prototypes to arrive at a final requirements specification in order for the rest of the design process to proceed.

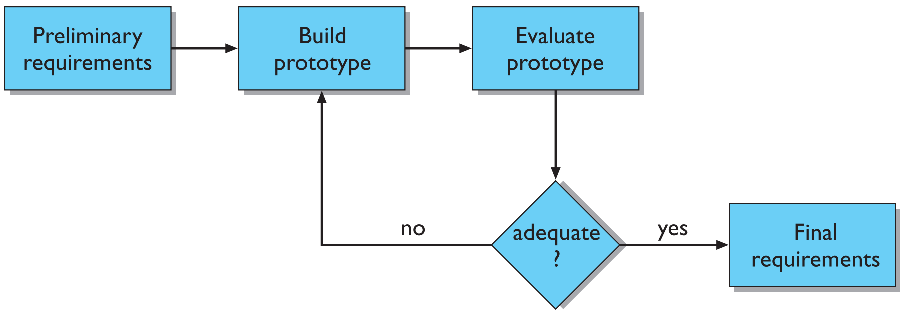


Figure 6.5 Throw-away prototyping within requirements specification

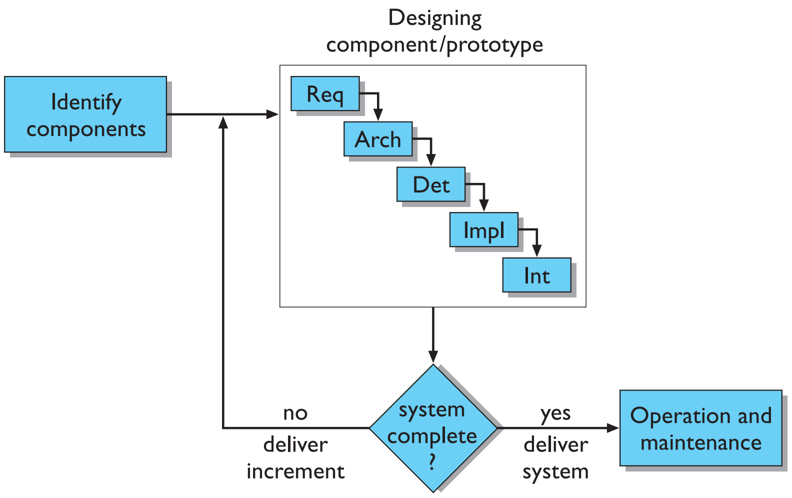


Figure 6.6 Incremental prototyping within the life cycle

Incremental The final product is built as separate components, one at a time. There is one overall design for the final system, but it is partitioned into independent and smaller components. The final product is then released as a series of products, each subsequent release including one more component. This is depicted in Figure 6.6.

Evolutionary Here the prototype is not discarded and serves as the basis for the next iteration of design. In this case, the actual system is seen as evolving from a very limited initial version to its final release, as depicted in Figure 6.7. Evolutionary prototyping also fits in well with the modifications which must be made to the system that arise during the operation and maintenance activity in the life cycle.

Prototypes differ according to the amount of functionality and performance they provide relative to the final product. An *animation* of requirements can involve no

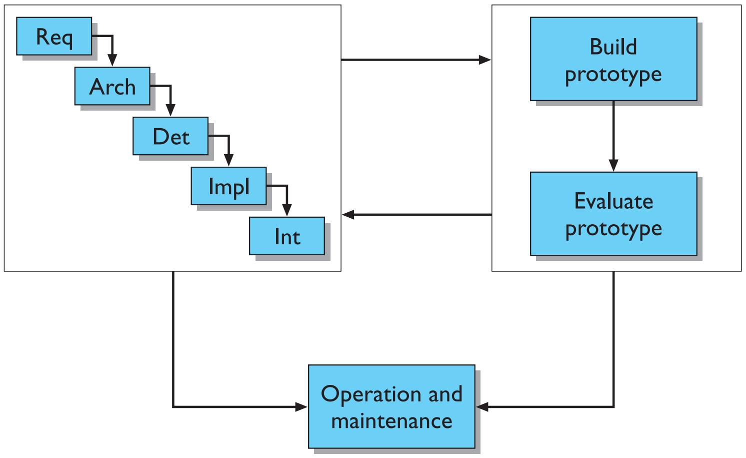


Figure 6.7 Evolutionary prototyping throughout the life cycle

real functionality, or limited functionality to simulate only a small aspect of the interactive behavior for evaluative purposes. At the other extreme, full functionality can be provided at the expense of other performance characteristics, such as speed or error tolerance. Regardless of the level of functionality, the importance of a prototype lies in its projected realism. The prototype of an interactive system is used to test requirements by evaluating their impact with real users. An honest appraisal of the requirements of the final system can only be trusted if the evaluation conditions are similar to those anticipated for the actual operation. But providing realism is costly, so there must be support for a designer/programmer to create a realistic prototype quickly and efficiently.

On the management side, there are several potential problems, as pointed out by Sommerville [327]:

Time Building prototypes takes time and, if it is a throw-away prototype, it can be seen as precious time taken away from the real design task. So the value of prototyping is only appreciated if it is fast, hence the use of the term *rapid prototyping*. However, rapid development and manipulation of a prototype should not be mistaken for rushed evaluation which might lead to erroneous results and invalidate the only advantage of using a prototype in the first place.

Planning Most project managers do not have the experience necessary for adequately planning and costing a design process which involves prototyping.

Non-functional features Often the most important features of a system will be non-functional ones, such as safety and reliability, and these are precisely the kinds of features which are sacrificed in developing a prototype. For evaluating usability features of a prototype, response time – yet another feature often compromised in a prototype – could be critical to product acceptance. This problem is similar to the technical issue of prototype realism.

Contracts The design process is often governed by contractual agreements between customer and designer which are affected by many of these managerial and technical issues. Prototypes and other implementations cannot form the basis for a legal contract, and so an iterative design process will still require documentation which serves as the binding agreement. There must be an effective way of translating the results derived from prototyping into adequate documentation. A rapid prototyping process might be amenable to quick changes, but that does not also apply to the design process.

6.4.1 Techniques for prototyping

Here we will describe some of the techniques that are available for producing rapid prototypes.

Storyboards

Probably the simplest notion of a prototype is the *storyboard*, which is a graphical depiction of the outward appearance of the intended system, without any accompanying system functionality. Storyboards do not require much in terms of computing power to construct; in fact, they can be mocked up without the aid of any computing resource. The origins of storyboards are in the film industry, where a series of panels roughly depicts snapshots from an intended film sequence in order to get the idea across about the eventual scene. Similarly, for interactive system design, the storyboards provide snapshots of the interface at particular points in the interaction. Evaluating customer or user impressions of the storyboards can determine relatively quickly if the design is heading in the right direction.

Modern graphical drawing packages now make it possible to create storyboards with the aid of a computer instead of by hand. Though the graphic design achievable on screen may not be as sophisticated as that possible by a professional graphic designer, it is more realistic because the final system will have to be displayed on a screen. Also, it is possible to provide crude but effective *animation* by automated sequencing through a series of snapshots. Animation illustrates the dynamic aspects of the intended user–system interaction, which may not be possible with traditional paper-based storyboards. If not animated, storyboards usually include annotations and scripts indicating how the interaction will occur.

Limited functionality simulations

More functionality must be built into the prototype to demonstrate the work that the application will accomplish. Storyboards and animation techniques are not sufficient for this purpose, as they cannot portray adequately the interactive aspects of the system. To do this, some portion of the functionality must be *simulated* by the design team.

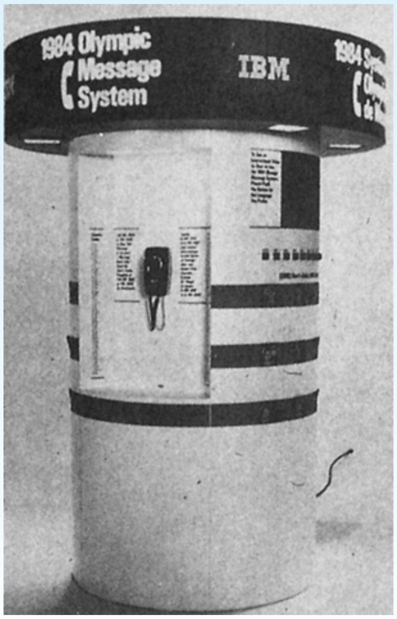
Programming support for simulations means a designer can rapidly build graphical and textual interaction objects and attach some behavior to those objects, which mimics the system's functionality. Once this simulation is built, it can be evaluated and changed rapidly to reflect the results of the evaluation study with various users.

For example, we might want to build a prototype for the VCR with undo described earlier using only a workstation display, keyboard and mouse. We could draw a picture of the VCR with its control panel using a graphics drawing package, but then we would want to allow a subject to use the mouse to position a finger cursor over one of the buttons to 'press' it and actuate some behavior of the VCR. In this way, we could simulate the programming task and experiment with different options for undoing.

Design Focus

Prototyping in practice

IBM supplied the computerized information and messaging booths for the 1984 Olympics in Los Angeles. These booths were to be used by the many thousands of residents in the Olympic village who would have to use them with no prior training (extensive instructions in several hundred languages being impractical). IBM sampled several variants on the kiosk design of the telephone-based system, using what they called the hallway and storefront methodology [152]. The final system was intended to be a walk-up-and-use system, so it was important to get comments from people with no knowledge of the process. Early versions of the kiosk were displayed as storyboards on a mock kiosk design in the front hallway of the Yorktown Research Lab. Passers-by were encouraged to browse at the display much as they would a storefront in the window. As casual comments were made and the kiosk was modified according to those comments, more and more active evaluation was elicited. This procedure helped to determine the ultimate positioning of display screens and telephones for the final design.



An Olympic Message System Kiosk (Gould J. D., Boies S. J., Levy S., Richards J. T. and Schoonard J. (1987). The 1984 Olympic Message System: a test of behavioral principles of system design. *Communications of the ACM*, **30**(9), 758–69. Copyright © 1987 ACM, Inc. Reprinted by permission)

There are now plenty of prototyping tools available which allow the rapid development of such simulation prototypes. These simulation tools are meant to provide a quick development process for a very wide range of small but highly interactive applications. A well-known and successful prototyping tool is *HyperCard*, a simulation environment for the Macintosh line of Apple computers. HyperCard is similar to the animation tools described above in that the user can create a graphical depiction of some system, say the VCR, with common graphical tools. The graphical images are placed on cards, and links between cards can be created which control the sequencing from one card to the next for animation effects. What HyperCard provides beyond this type of animation is the ability to describe more sophisticated interactive behavior by attaching a *script*, written in the HyperTalk programming language, to any object. So for the VCR, we could attach a script to any control panel button to highlight it or make an audible noise when the user clicks the mouse cursor over it. Then some functionality could be associated to that button by reflecting some change in the VCR display window. Similar functionality is provided through tools such as Macromedia Flash and Director.

Most of the simulations produced are intended to be throw-away prototypes because of their relatively inefficient implementation. They are not intended to support full-blown systems development and they are unsatisfactory in that role. However, as more designers recognize the utility of prototyping and iterative design, they are beginning to demand ways of incorporating the prototypes into the final delivered systems – more along the lines of evolutionary prototyping. A good example of this is in the avionics industry, where it has long been recognized that iterative development via rapid prototyping and evaluation is essential for the design of flight deck instrumentation and controls. Workstation technology provides sufficient graphics capabilities to enable a designer to produce very realistic gauges, which can be assessed and critiqued by actual pilots. With the advent of the glass cockpit – in which traditional mechanical gauges are replaced by gauges represented on video displays – there is no longer a technology gap between the prototype designs of flight deck instruments and the actual instruments in flight. Therefore, it is a reasonable request by these designers that they be able to reuse the functionality of the prototypes in the actual flight simulators and cockpits, and this demand is starting to be met by commercial prototyping systems which produce efficient code for use in such safety-critical applications.

One technique for simulation, which does not require very much computer-supported functionality, is the *Wizard of Oz* technique. With this technique, the designers can develop a limited functionality prototype and enhance its functionality in evaluation by providing the missing functionality through human intervention. A participant in the evaluation of a new accounting system may not have any computer training but is familiar with accounting procedures. He is asked to sit down in front of the prototype accounting system and to perform some task, say to check the accounts receivable against some newly arrived payments. The naïve computer user will not know the specific language of the system, but you do not want him to worry about that. Instead, he is given instructions to type whatever seems the most natural commands to the system. One of the designers – the wizard in this scenario – is situated in another room, out of sight of the subject, but she is able to receive the subject's input commands and translate them into commands that will work on the prototype. By intervening between the user and system, the wizard is able to increase the perceived functionality of the system so that evaluation can concentrate on how the subject would react to the complete system. Examination of how the wizard had to interpret the subject's input can provide advice as to how the prototype must be enhanced in its later versions.

High-level programming support

HyperTalk was an example of a special-purpose high-level programming language which makes it easy for the designer to program certain features of an interactive system at the expense of other system features like speed of response or space efficiency. HyperTalk and many similar languages allow the programmer to attach functional behavior to the specific interactions that the user will be able to do, such as position and click on the mouse over a button on the screen. Previously, the difficulty of interactive programming was that it was so implementation dependent that the programmer would have to know quite a bit of intimate detail of the hardware system in order to control even the simplest of interactive behavior. These high-level programming languages allow the programmer to abstract away from the hardware specifics and think in terms that are closer to the way the input and output devices are perceived as interaction devices.

Though not usually considered together with such simulation environments, a *user interface management system* – or UIMS (pronounced 'you-imz') – can be considered to provide such high-level programming support. The frequent conceptual model put forth for interactive system design is to separate the application functionality from its presentation. It is then possible to program the underlying functionality of the system and to program the behavior of the user interface separately. The job of a UIMS, then, is to allow the programmer to connect the behavior at the interface with the underlying functionality. In Chapter 8 we will discuss in more detail the advantages and disadvantages of such a conceptual model and concentrate on the programming implementation support provided by a UIMS. What is of interest here is that the separation implied by a UIMS allows the independent development of the features of the interface apart from the underlying functionality. If the underlying system is already developed, then various prototypes of its interface can be quickly constructed and evaluated to determine the optimal one.

6.4.2 Warning about iterative design

Though we have presented the process of iterative design as not only beneficial but also necessary for good interactive system design, it is important to recognize some of its drawbacks, in addition to the very real management issues we have already raised. The ideal model of iterative design, in which a rapid prototype is designed, evaluated and modified until the best possible design is achieved in the given project time, is appealing. But there are two problems.

First, it is often the case that design decisions made at the very beginning of the prototyping process are wrong and, in practice, design inertia can be so great as never to overcome an initial bad decision. So, whereas iterative design is, in theory, amenable to great changes through iterations, it can be the case that the initial prototype has bad features that will not be amended. We will examine this problem through a real example of a clock on a microwave oven.[2] The clock has a numeric display of four digits. Thus the display is capable of showing values in the range from `00:00` to `99:99`. The functional model of time for the actual clock is only 12 hours, so quite a few of the possible clock displays do not correspond to possible times (for example, `63:00`, `85:49`), even though some of them are legal four-digit time designations. That poses no problem, as long as both the designer and the ultimate users of the clock both share the knowledge of the discrepancy between possible clock displays and legal times. Such would not be the case for someone assuming a 24-hour time format, in which case the displays `00:30` and `13:45` would represent valid times in their model but not in the microwave's model. In this particular example, the subjects tested during the evaluation must have all shared the 12-hour time model, and the mismatch with the other users (with a 24-hour model) was only discovered after the product was being shipped. At this point, the only impact of iterative design was a change to the documentation alerting the reader to the 12-hour format, as it was too late to perform any hardware change.

The second problem is slightly more subtle, and serious. If, in the process of evaluation, a potential usability problem is diagnosed, it is important to understand the reason for the problem and not just detect the symptom. In the clock example, the designers could have noticed that some subjects with a 24-hour time model were having difficulty setting the time. Say they were trying to set the time for `14:45`, but they were not being allowed to do that. If the designers did not know the subject's goals, they might not detect the 24/12 hour discrepancy. They would instead notice that the users were having trouble setting the time and so they might change the buttons used to set the time instead of other possible changes, such as an analog time dial, or displaying AM or PM on the clock dial to make the 12-hour model more obvious, or to change to a 24-hour clock.

The moral for iterative design is that it should be used in conjunction with other, more principled approaches to interactive system design. These principled approaches are the subject of Part 3 of this book.

6.5 DESIGN RATIONALE

In designing any computer system, many decisions are made as the product goes from a set of vague customer requirements to a deliverable entity. Often it is difficult to recreate the reasons, or rationale, behind various design decisions. *Design*

2 This example has been provided by Harold Thimbleby.

rationale is the information that explains why a computer system is the way it is, including its structural or architectural description and its functional or behavioral description. In this sense, design rationale does not fit squarely into the software life cycle described in this chapter as just another phase or box. Rather, design rationale relates to an activity of both reflection (doing design rationale) and documentation (creating a design rationale) that occurs throughout the entire life cycle.

It is beneficial to have access to the design rationale for several reasons:

- In an explicit form, a design rationale provides a communication mechanism among the members of a design team so that during later stages of design and/or maintenance it is possible to understand what critical decisions were made, what alternatives were investigated (and, possibly, in what order) and the reason why one alternative was chosen over the others. This can help avoid incorrect assumptions later.
- Accumulated knowledge in the form of design rationales for a set of products can be reused to transfer what has worked in one situation to another situation which has similar needs. The design rationale can capture the context of a design decision in order that a different design team can determine if a similar rationale is appropriate for their product.
- The effort required to produce a design rationale forces the designer to deliberate more carefully about design decisions. The process of deliberation can be assisted by the design rationale technique by suggesting how arguments justifying or discarding a particular design option are formed.

In the area of HCI, design rationale has been particularly important, again for several reasons:

- There is usually no single best design alternative. More often, the designer is faced with a set of trade-offs between different alternatives. For example, a graphical interface may involve a set of actions that the user can invoke by use of the mouse and the designer must decide whether to present each action as a 'button' on the screen, which is always visible, or hide all of the actions in a menu which must be explicitly invoked before an action can be chosen. The former option maximizes the operation visibility (see Chapter 7) but the latter option takes up less screen space. It would be up to the designer to determine which criterion for evaluating the options was more important and then communicating that information in a design rationale.
- Even if an optimal solution did exist for a given design decision, the space of alternatives is so vast that it is unlikely a designer would discover it. In this case, it is important that the designer indicates all alternatives that have been investigated. Then later on it can be determined if she has not considered the best solution or had thought about it and discarded it for some reason. In project management, this kind of accountability for design is good.
- The usability of an interactive system is very dependent on the context of its use. The flashiest graphical interface is of no use if the end-user does not have access to a high-quality graphics display or a pointing device. Capturing the context in

which a design decision is made will help later when new products are designed. If the context remains the same, then the old rationale can be adopted without revision. If the context has changed somehow, the old rationale can be reexamined to see if any rejected alternatives are now more favorable or if any new alternatives are now possible.

Lee and Lai [209] explain that various proponents of design rationale have different interpretations of what it actually is. We will make use of their classification to describe various design rationale techniques in this section. The first set of techniques concentrates on providing a historical record of design decisions and is very much tailored for use during actual design discussions. These techniques are referred to as process-oriented design rationale because they are meant to be integrated in the actual design process itself. The next category is not so concerned with historical or process-oriented information but rather with the structure of the space of all design alternatives, which can be reconstructed by post hoc consideration of the design activity. The structure-oriented approach does not capture historical information. Instead, it captures the complete story of the moment, as an analysis of the design space which has been considered so far. The final category of design rationale concentrates on capturing the claims about the psychology of the user that are implied by an interactive system and the tasks that are performed on them.

There are some issues that distinguish the various techniques in terms of their usability within design itself. We can use these issues to sketch an informal rationale for design rationale. One issue is the degree to which the technique impinges on the design process. Does the use of a particular design rationale technique alter the decision process, or does it just passively serve to document it? Another issue is the cost of using the technique, both in terms of creating the design rationale and in terms of accessing it once created. A related issue is the amount of computational power the design rationale provides and the level to which this is supported by automated tools. A design rationale for a complex system can be very large and the exploration of the design space changes over time. The kind of information stored in a given design rationale will affect how that vast amount of information can be effectively managed and browsed.

6.5.1 Process-oriented design rationale

Much of the work on design rationale is based on Rittel's *issue-based information system*, or *IBIS*, a style for representing design and planning dialog developed in the 1970s [308]. In IBIS (pronounced 'ibbiss'), a hierarchical structure to a design rationale is created. A root *issue* is identified which represents the main problem or question that the argument is addressing. Various *positions* are put forth as potential resolutions for the root issue, and these are depicted as descendants in the IBIS hierarchy directly connected to the root issue. Each position is then supported or refuted by *arguments*, which modify the relationship between issue and position. The hierarchy grows as secondary issues are raised which modify the root issue in some way. Each of these secondary issues is in turn expanded by positions and arguments, further sub-issues, and so on.

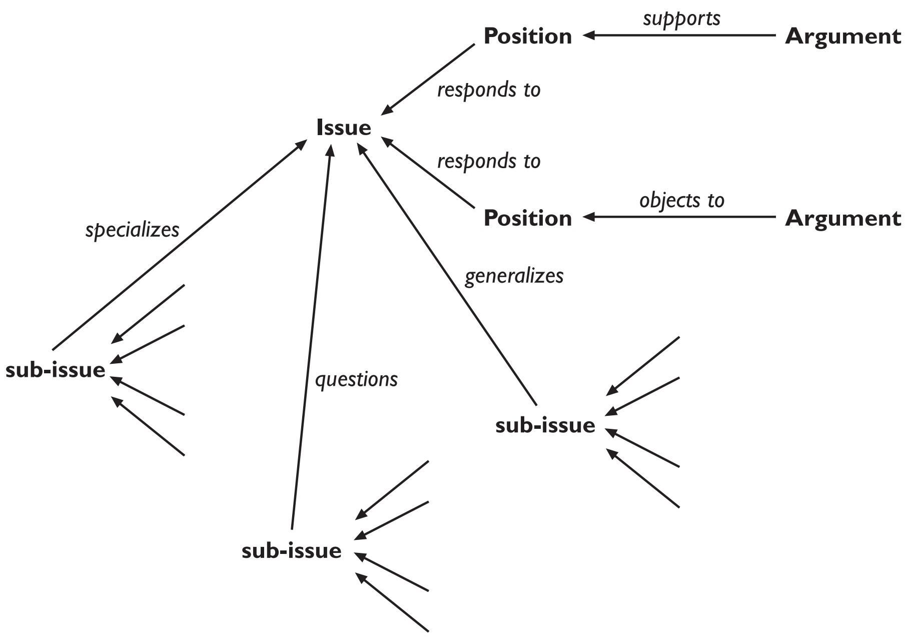


Figure 6.8 The structure of a gIBIS design rationale

A graphical version of IBIS has been defined by Conklin and Yakemovic [77], called *gIBIS* (pronounced 'gibbiss'), which makes the structure of the design rationale more apparent visually in the form of a directed graph which can be directly edited by the creator of the design rationale. Figure 6.8 gives a representation of the gIBIS vocabulary. Issues, positions and arguments are nodes in the graph and the connections between them are labeled to clarify the relationship between adjacent nodes. So, for example, an issue can suggest further sub-issues, or a position can respond to an issue or an argument can support a position. The gIBIS structure can be supported by a hypertext tool to allow a designer to create and browse various parts of the design rationale.

There have been other versions of the IBIS notation, both graphical and textual, besides gIBIS. Most versions retain the distinction between issues, positions and arguments. Some add further nodes, such as Potts and Bruns's [297] addition of design artifacts which represent the intermediate products of a design that lead to the final product and are associated with the various alternatives discussed in the design rationale. Some add a richer vocabulary to modify the relationships between the node elements, such as McCall's Procedural Hierarchy of Issues (PHI) [231], which expands the variety of inter-issue relationships. Interesting work at the University of Colorado has attempted to link PHI argumentation to computer-aided design (CAD) tools to allow critique of design (in their example, the design of a kitchen) as it occurs. When the CAD violates some known design rule, the designer is warned and can then browse a PHI argument to see the rationale for the design rule.

The use of IBIS and any of its descendants is process oriented, as we described above. It is intended for use during design meetings as a means of recording and structuring the issues deliberated and the decisions made. It is also intended to preserve the order of deliberation and decision making for a particular product, placing less stress on the generalization of design knowledge for use between different products. This can be contrasted with the structure-oriented technique discussed next.

6.5.2 Design space analysis

MacLean and colleagues [222] have proposed a more deliberative approach to design rationale which emphasizes a post hoc structuring of the space of design alternatives that have been considered in a design project. Their approach, embodied in the Questions, Options and Criteria (QOC) notation, is characterized as *design space analysis* (see Figure 6.9).

The design space is initially structured by a set of questions representing the major issues of the design. Since design space analysis is structure oriented, it is not so important that the questions recorded are the actual questions asked during design meetings. Rather, these questions represent an agreed characterization of the

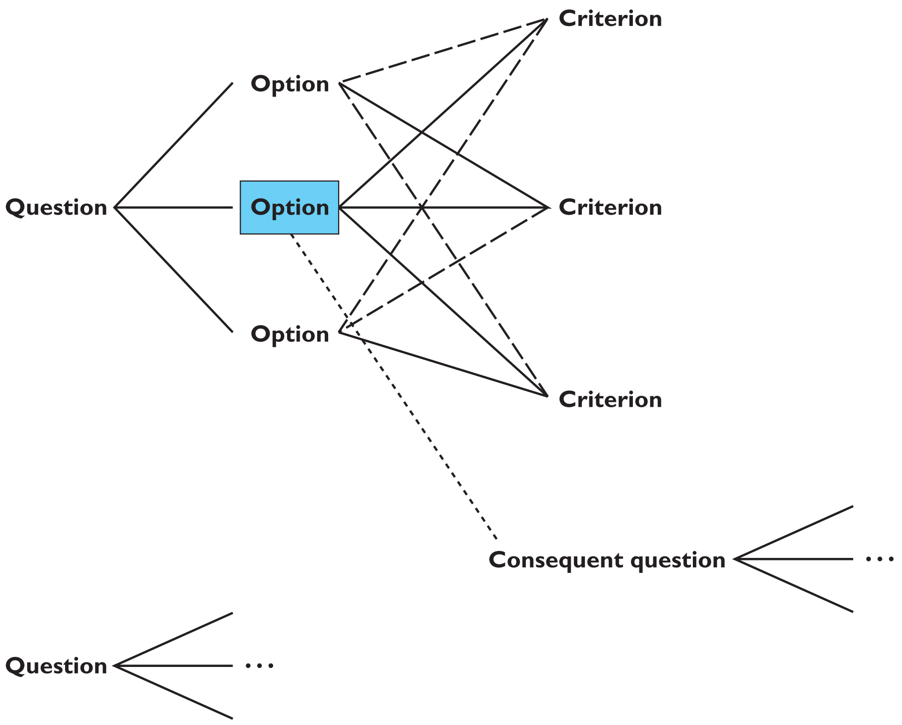


Figure 6.9 The QOC notation

issues raised based on reflection and understanding of the actual design activities. Questions in a design space analysis are therefore similar to issues in IBIS except in the way they are captured. Options provide alternative solutions to the question. They are assessed according to some criteria in order to determine the most favorable option. In Figure 6.9 an option which is favorably assessed in terms of a criterion is linked with a solid line, whereas negative links have a dashed line. The most favorable option is boxed in the diagram.

The key to an effective design space analysis using the QOC notation is deciding the right questions to use to structure the space and the correct criteria to judge the options. The initial questions raised must be sufficiently general that they cover a large enough portion of the possible design space, but specific enough that a range of options can be clearly identified. It can be difficult to decide the right set of criteria with which to assess the options. The QOC technique advocates the use of general criteria, like the usability principles we shall discuss in Chapter 7, which are expressed more explicitly in a given analysis. In the example of the action buttons versus the menu of actions described earlier, we could contextualize the general principle of operation visibility as the criterion that all possible actions are displayed at all times. It can be very difficult to decide from a design space analysis which option is most favorable. The positive and negative links in the QOC notation do not provide all of the context for a trade-off decision. There is no provision for indicating, for example, that one criterion is more important than any of the others and the most favorable option must be positively linked.

Another structure-oriented technique, called Decision Representation Language (DRL), developed by Lee and Lai, structures the design space in a similar fashion to QOC, though its language is somewhat larger and it has a formal semantics. The questions, options and criteria in DRL are given the names: decision problem, alternatives and goals. QOC assessments are represented in DRL by a more complex language for relating goals to alternatives. The sparse language in QOC used to assess an option relative to a criterion (positive or negative assessment only) is probably insufficient, but there is a trade-off involved in adopting a more complex vocabulary which may prove too difficult to use in practice. The advantage of the formal semantics of DRL is that the design rationale can be used as a computational mechanism to help manage the large volume of information. For example, DRL can track the dependencies between different decision problems, so that subsequent changes to the design rationale for one decision problem can be automatically propagated to other dependent problems.

Design space analysis directly addresses the claim that no design activity can hope to uncover all design possibilities, so the best we can hope to achieve is to document the small part of the design space that has been investigated. An advantage of the post hoc technique is that it can abstract away from the particulars of a design meeting and therefore represent the design knowledge in such a way that it can be of use in the design of other products. The major disadvantage is the increased overhead such an analysis warrants. More time must be taken away from the design activity to do this separate documentation task. When time is scarce, these kinds of overhead costs are the first to be trimmed.

6.5.3 Psychological design rationale

The final category of design rationale tries to make explicit the psychological claims of usability inherent in any interactive system in order better to suit a product for the tasks users have. This psychological design rationale has been introduced by Carroll and Rosson [62], and before we describe the application of the technique it is important to understand some of its theoretical background.

People use computers to accomplish some tasks in their particular work domain, as we have seen before. When designing a new interactive system, the designers take into account the tasks that users currently perform and any new ones that they may want to perform. This task identification serves as part of the requirements for the new system, and can be done through empirical observation of how people perform their work currently and presented through informal language or a more formal task analysis language (see Chapter 15). When the new system is implemented, or becomes an *artifact*, further observation reveals that in addition to the required tasks it was built to support, it also supports users in tasks that the designer never intended. Once designers understand these new tasks, and the associated problems that arise between them and the previously known tasks, the new task definitions can serve as requirements for future artifacts.

Carroll refers to this real-life phenomenon as the *task–artifact cycle*. He provides a good example of this cycle through the evolution of the electronic spreadsheet. When the first electronic spreadsheet, *VisiCalc*, was marketed in the late 1970s, it was presented simply as an automated means of supporting tabular calculation, a task commonly used in the accounting world. Within little over a decade of its introduction, the application of spreadsheets had far outstripped its original intent within accounting. Spreadsheets were being used for all kinds of financial analysis, 'what-if' simulations, report formatting and even as a general programming language paradigm! As the set of tasks expands, new spreadsheet products have flooded the marketplace trying to satisfy the growing customer base. Another good example of the task–artifact cycle in action is with word processing, which was originally introduced to provide more automated support for tasks previously achieved with a typewriter and now provides users with the ability to carry out various authoring tasks that they never dreamed possible with a conventional typewriter. And today, the tasks for the spreadsheet and the word processor are intermingled in the same artifact.

The purpose of psychological design rationale is to support this natural task–artifact cycle of design activity. The main emphasis is not to capture the designer's intention in building the artifact. Rather, psychological design rationale aims to make explicit the consequences of a design for the user, given an understanding of what tasks he intends to perform. Previously, these psychological consequences were left implicit in the design, though designers would make informal claims about their systems (for example, that it is more 'natural' for the user, or easier to learn).

The first step in the psychological design rationale is to identify the tasks that the proposed system will address and to characterize those tasks by questions that the user tries to answer in accomplishing them. For instance, Carroll gives an example of designing a system to help programmers learn the Smalltalk object-oriented programming language environment. The main task the system is to support is learning how Smalltalk works. In learning about the programming environment, the programmer will perform tasks that help her answer the questions:

- What can I do: that is, what are the possible operations or functions that this programming environment allows?
- How does it work: that is, what do the various functions do?
- How can I do this: that is, once I know a particular operation I want to perform, how do I go about programming it?

For each question, a set of *scenarios* of user–system behavior is suggested to support the user in addressing the question. For example, to address the question 'What can I do?', the designers can describe a scenario whereby the novice programmer is first confronted with the learning environment and sees that she can invoke some demo programs to investigate how Smalltalk programs work. The initial system can then be implemented to provide the functionality suggested by the scenarios (for example, some demos would be made accessible and obvious to the user/programmer from the very beginning). Once this system is running, observation of its use and some designer reflection is used to produce the actual psychological design rationale for that version of the system. This is where the psychological claims are made explicit. For example, there is an assumption that the programmer knows that what she can see on the screen relates to what she can do (if she sees the list of programs under a heading 'Demos', she can click on one program name to see the associated demo). The psychological claim of this demo system is that the user learns by doing, which is a good thing. However, there may also be negative aspects that are equally important to mention. The demo may not be very interactive, in which case the user clicks on it to initiate it and then just sits back and watches a graphic display, never really learning how the demo application is constructed in Smalltalk. These negative aspects can be used to modify later versions of the system to allow more interactive demos, which represent realistic, yet simple, applications, whose behavior and structure the programmer can appreciate.

By forcing the designer to document the psychological design rationale, it is hoped that she will become more aware of the natural evolution of user tasks and the artifact, taking advantage of how consequences of one design can be used to improve later designs.

Worked exercise *What is the distinction between a process-oriented and a structure-oriented design rationale technique? Would you classify psychological design rationale as process or structure oriented? Why?*

Answer The distinction between a process- and structure-oriented design rationale resides in what information the design rationale attempts to capture. Process-oriented design rationale is interested in recording an historically accurate description of a design team making some decision on a particular issue for the design. In this sense, process-oriented design rationale becomes an activity concurrent with the rest of the design process. Structure-oriented design rationale is less interested in preserving the historical evolution of the design. Rather, it is more interested in providing the conclusions of the design activity, so it can be done in a post hoc and reflective manner after the fact.

The purpose of psychological design rationale is to support the task–artifact cycle. Here, the tasks that the users perform are changed by the systems on which they perform the tasks. A psychological design rationale proceeds by having the designers of the system record what they believe are the tasks that the system should support and then building the system to support the tasks. The designers suggest scenarios for the tasks which will be used to observe new users of the system. Observations of the users provide the information needed for the actual design rationale of that version of the system. The consequences of the design's assumptions about the important tasks are then gauged against the actual use in an attempt to justify the design or suggest improvements.

Psychological design rationale is mainly a process-oriented approach. The activity of a claims analysis is precisely about capturing what the designers assumed about the system at one point in time and how those assumptions compared with actual use. Therefore, the history of the psychological design rationale is important. The discipline involved in performing a psychological design rationale requires designers to perform the claims analysis during the actual design activity, and not as post hoc reconstruction.

6.6 SUMMARY

In this chapter, we have shown how software engineering and the design process relate to interactive system design. The software engineering life cycle aims to structure design in order to increase the reliability of the design process. For interactive system design, this would equate to a reliable and reproducible means of designing predictably usable systems. Because of the special needs of interactive systems, it is essential to augment the standard life cycle in order to address issues of HCI.

Usability engineering encourages incorporating explicit usability goals within the design process, providing a means by which the product's usability can be judged. Iterative design practices admit that principled design of interactive systems alone cannot maximize product usability, so the designer must be able to evaluate early prototypes and rapidly correct features of the prototype which detract from the product usability.

The design process is composed of a series of decisions, which pare down the vast set of potential systems to the one that is actually delivered to the customer. Design rationale, in its many forms, is aimed at allowing the designer to manage the information about the decision-making process, in terms of when and why design decisions were made and what consequences those decisions had for the user in accomplishing his work.

EXERCISES

6.1 (a) How can design rationale benefit interface design and why might it be rejected by design teams?
(b) Explain QOC design rationale using an example to illustrate.

6.2 Imagine you have been asked to produce a prototype for the diary system discussed in the worked exercise in Section 7.2.3. What would be an appropriate prototyping approach to enable you to test the design using the usability metrics specified, and why?

RECOMMENDED READING

J. A. McDermid, editor, *The Software Engineer's Reference Book*, Butterworth–Heinemann, 1992.
A very good general reference book for all topics in software engineering. In particular, we refer you to Chapter 15 on software life cycles and Chapter 40 on prototyping.

I. Sommerville, *Software Engineering*, 6th edition, Addison-Wesley, 2000.
This textbook is one of the few texts in software engineering that specifically treats issues of interface design.

X. Faulkner, *Usability Engineering*, Macmillan, 2000.
An excellent and accessible introduction to usability engineering covering, amongst other things, user requirements capture and usability metrics.

J. Whiteside, J. Bennett and K. Holtzblatt, Usability engineering: our experience and evolution. In M. Helander, editor, *Handbook for Human–Computer Interaction*, North-Holland, 1988.
The seminal work on usability engineering. More recent work on usability engineering has also been published by Jakob Nielsen [260, 261].

J. M. Carroll and T. P. Moran, editors, *Design Rationale: Concepts, Techniques and Use*, Lawrence Erlbaum, 1996.
Expanded from a double special journal issue, this provides comprehensive coverage of relevant work in the field.

7 Design rules

Overview

- Designing for maximum usability is the goal of interactive systems design.
- Abstract principles offer a way of understanding usability in a more general sense, especially if we can express them within some coherent catalog.
- Design rules in the form of standards and guidelines provide direction for design, in both general and more concrete terms, in order to enhance the interactive properties of the system.
- The essential characteristics of good design are often summarized through ‘golden rules’ or heuristics.
- Design patterns provide a potentially generative approach to capturing and reusing design knowledge.

7.1 INTRODUCTION

One of the central problems that must be solved in a user-centered design process is how to provide designers with the ability to determine the usability consequences of their design decisions. We require *design rules*, which are rules a designer can follow in order to increase the usability of the eventual software product. We can classify these rules along two dimensions, based on the rule's authority and generality. By authority, we mean an indication of whether or not the rule must be followed in design or whether it is only suggested. By generality, we mean whether the rule can be applied to many design situations or whether it is focussed on a more limited application situation. Rules also vary in their level of abstraction, with some abstracting away from the detail of the design solution and others being quite specific. It is also important to determine the origins of a design rule. We will consider a number of different types of design rules. *Principles* are abstract design rules, with high generality and low authority. *Standards* are specific design rules, high in authority and limited in application, whereas *guidelines* tend to be lower in authority and more general in application.

Design rules for interactive systems can be supported by psychological, cognitive, ergonomic, sociological, economic or computational theory, which may or may not have roots in empirical evidence. Designers do not always have the relevant background in psychology, cognitive science, ergonomics, sociology, business or computer science necessary to understand the consequences of those theories in the instance of the design they are creating. The design rules are used to apply the theory in practice. Often a set of design rules will be in conflict with each other, meaning that strict adherence to all of them is impossible. The theory underlying the separate design rules can help the designer understand the trade-off for the design that would result in following or disregarding some of the rules. Usually, the more general a design rule is, the greater the likelihood that it will conflict with other rules and the greater the need for the designer to understand the theory behind it.

We can make another rough distinction between principles, standards and guidelines. Principles are derived from knowledge of the psychological, computational and sociological aspects of the problem domains and are largely independent of the technology; they depend to a much greater extent on a deeper understanding of the human element in the interaction. They can therefore be applied widely but are not so useful for specific design advice. Guidelines are less abstract and often more technology oriented, but as they are also general, it is important for a designer to know what theoretical evidence there is to support them. A designer will have less of a need to know the underlying theory for applying a standard. However, since standards carry a much higher level of authority, it is more important that the theory underlying them be correct or sound.

The previous chapter was about the process of design, and we need to consider when design rules can be of use within that process. Design rules are mechanisms for restricting the space of design options, preventing a designer from pursuing design options that would be likely to lead to an unusable system. Thus, design rules would be most effective if they could be adopted in the earliest stages of the life cycle, such as in requirements specification and architectural design, when the space of possible designs is still very large. We have already seen, for example, in Chapter 6, how abstract principles can be applied in usability engineering.

However, if the designer does not understand the assumptions underlying a design rule, it is quite possible that early application can prevent the best design choice. For example, a set of design rules might be specific to a particular hardware platform and inappropriate for other platforms (for example, color versus monochrome screens, one- versus two- or three-button mouse). Such bias in some design rules causes them to be applicable only in later stages of the life cycle.

We will first discuss abstract principles, then go on to consider in more depth some examples of standards and guidelines for user-centered design. Finally, we will consider some well-known heuristics or 'golden rules' which, it has been suggested, provide a succinct summary of the essence of good design. We end the chapter with a discussion of design patterns, a relatively new approach to capturing design knowledge in HCI.

7.2 PRINCIPLES TO SUPPORT USABILITY

The most abstract design rules are general principles, which can be applied to the design of an interactive system in order to promote its usability. In Chapter 4 we looked at the different paradigms that represent the development of interactive systems. Derivation of principles for interaction has usually arisen out of a need to explain why a paradigm is successful and when it might not be. Principles can provide the repeatability which paradigms in themselves cannot provide. In this section we present a collection of usability principles. Since it is too bold an objective to produce a comprehensive catalog of such principles, our emphasis will be on structuring the presentation of usability principles in such a way that the catalog can be easily extended as our knowledge increases.

The principles we present are first divided into three main categories:

Learnability – the ease with which new users can begin effective interaction and achieve maximal performance.

Flexibility – the multiplicity of ways in which the user and system exchange information.

Robustness – the level of support provided to the user in determining successful achievement and assessment of goals.

In the following, we will subdivide these main categories into more specific principles that support them. In most cases, we are able to situate these more specific principles within a single category, but we have made explicit those cases when a principle falls into two of the above categories.

Table 7.1 Summary of principles affecting learnability

Principle	Definition	Related principles
Predictability	Support for the user to determine the effect of future action based on past interaction history	Operation visibility
Synthesizability	Support for the user to assess the effect of past operations on the current state	Immediate/eventual honesty
Familiarity	The extent to which a user's knowledge and experience in other real-world or computer-based domains can be applied when interacting with a new system	Guessability, affordance
Generalizability	Support for the user to extend knowledge of specific interaction within and across applications to other similar situations	–
Consistency	Likeness in input–output behavior arising from similar situations or similar task objectives	–

7.2.1 Learnability

Learnability concerns the features of the interactive system that allow novice users to understand how to use it initially and then how to attain a maximal level of performance. Table 7.1 contains a summary of the specific principles that support learnability, which we will describe below.

Predictability

Except when interacting with some video games, a user does not take very well to surprises. Predictability of an interactive system means that the user's knowledge of the interaction history is sufficient to determine the result of his future interaction with it. There are many degrees to which predictability can be satisfied. The knowledge can be restricted to the presently perceivable information, so that the user need not remember anything other than what is currently observable. The knowledge requirement can be increased to the limit where the user is actually forced to remember what every previous keystroke was and what every previous screen display contained (and the order of each!) in order to determine the consequences of the next input action.

Predictability of an interactive system is distinguished from deterministic behavior of the computer system alone. Most computer systems are ultimately deterministic machines, so that given the state at any one point in time and the operation which is to be performed at that time, there is only one possible state that can result. Predictability is a user-centered concept; it is deterministic behavior from the perspective of the user. It is not enough for the behavior of the computer system to be determined completely from its state, as the user must be able to take advantage of the determinism.

For example, a common mathematical puzzle would be to present you with a sequence of three or more numbers and ask you what would be the next number in the sequence. The assumption in this puzzle (and one that can often be incorrect) is that there is a unique function or algorithm that produces the entire sequence of numbers and it is up you to figure it out. We know the function, but all you know are the results it provides from the first three calculations. The function is certainly deterministic; the test for you is a test of its predictability given the first three numbers in the sequence.

As another, possibly more pertinent, example, imagine you have created a complex picture using a mouse-driven graphical drawing package. You leave the picture for a few days and then go back to change it around a bit. You are allowed to select certain objects for editing by positioning the mouse over the object and clicking a mouse button to highlight it. Can you tell what the set of selectable objects is? Can you determine which area of the screen belongs to which of these objects, especially if some objects overlap? Does the visual image on the screen indicate what objects form a compound object that can only be selected as a group? Predictability of selection in this example depends on how much of the history of the creation of the visual image is necessary in order for you to determine what happens when you click on the mouse button.

This notion of predictability deals with the user's ability to determine the effect of operations on the system. Another form of predictability has to do with the user's ability to know which operations can be performed. *Operation visibility* refers to how the user is shown the availability of operations that can be performed next. If an operation can be performed, then there may be some perceivable indication of this to the user. This principle supports the superiority in humans of recognition over recall. Without it, the user will have to remember when he can perform the operation and when he cannot. Likewise, the user should understand from the interface if an operation he might like to invoke cannot be performed.

Synthesizability

Predictability focusses on the user's ability to determine the effect of future interactions. This assumes that the user has some mental model (see Chapter 1) of how the system behaves. Predictability says nothing about the way the user forms a model of the system's behavior. In building up some sort of predictive model of the system's behavior, it is important for the user to assess the consequences of previous interactions in order to formulate a model of the behavior of the system. Synthesis, therefore, is the ability of the user to assess the effect of past operations on the current state.

When an operation changes some aspect of the internal state, it is important that the change is seen by the user. The principle of *honesty* relates to the ability of the user interface to provide an observable and informative account of such change. In the best of circumstances, this notification can come *immediately*, requiring no further interaction initiated by the user. At the very least, the notification should appear *eventually*, after explicit user directives to make the change observable. A good example of the distinction between immediacy and eventuality can be seen in the comparison between command language interfaces and visual desktop interfaces for a file management system. You have moved a file from one directory to another. The principle of honesty implies that after moving the file to its new location in the file system you are then able to determine its new whereabouts. In a command language system, you would typically have to remember the destination directory and then ask to see the contents of that directory in order to verify that the file has been moved (in fact, you would also have to check that the file is no longer in its original directory to determine that it has been moved and not copied). In a visual desktop interface, a visual representation (or icon) of the file is dragged from its original directory and placed in its destination directory where it remains visible (assuming the destination folder is selected to reveal its contents). In this case, the user need not expend any more effort to assess the result of the move operation. The visual desktop is immediately honest.

The problem with eventual honesty is that the user must know to look for the change. In a situation in which the user is learning a new interactive system, it is likely that he will not know to look for change. In earlier versions of the Apple Macintosh Finder, performing the operation to create a new folder in another folder did not necessarily result in that new folder's icon being visible in the original folder. New users (and even some experienced users) would often think that they had not issued the new folder operations correctly and would ask for another new folder (and another, and another, . . .). They would not know to search through the entire open folder for the latest addition. Then several minutes (hours, days) later, they would notice that there were a number of empty and untitled folders lying around. The eventual (accidental) discovery of the change brought about by the new folder operation was then difficult to associate to that operation. Fortunately, this problem was addressed in Version 7 of the Finder.

As another example of the benefit of immediate over eventual honesty, let us examine a typical global search and replace function in a word processor. Imagine you have noticed in the past a tendency to repeat words in a document (for example, you type 'the the' without noticing the error). In an attempt to automate your proofreading, you decide to replace globally all occurrences of 'the the' with 'the'. The typical global search and replace function performs this substitution without revealing the changes made to you. Suddenly, a careless typing error is transformed into unacceptable grammar as the sentence

> We will prove the theorem holds as a corollary of the following lemma.

is transformed to

> We will prove theorem holds as a corollary of the following lemma.

Familiarity

New users of a system bring with them a wealth of experience across a wide number of application domains. This experience is obtained both through interaction in the real world and through interaction with other computer systems. For a new user, the familiarity of an interactive system measures the correlation between the user's existing knowledge and the knowledge required for effective interaction. For example, when word processors were originally introduced the analogy between the word processor and a typewriter was intended to make the new technology more immediately accessible to those who had little experience with the former but a lot of experience with the latter. Familiarity has to do with a user's first impression of the system. In this case, we are interested in how the system is first perceived and whether the user can determine how to initiate any interaction. An advantage of a metaphor, such as the typewriter metaphor for word processing described above, is precisely captured by familiarity. Jordan et al. refer to this familiarity as the *guessability* of the system [196].

Some psychologists argue that there are intrinsic properties, or *affordances*, of any visual object that suggest to us how they can be manipulated (see also Chapter 5, Section 5.7.2). The appearance of the object stimulates a familiarity with its behavior. For example, the shape of a door handle can suggest how it should be manipulated to open a door, and a key on a keyboard suggests to us that it can be pushed. In the design of a graphical user interface, it is implied that a soft button used in a form's interface suggests it should be pushed (though it does not suggest how it is to be pushed via the mouse). Effective use of the affordances that exist for interface objects can enhance the familiarity of the interactive system.

Generalizability

Users often try to extend their knowledge of specific interaction behavior to situations that are similar but previously unencountered. The generalizability of an interactive system supports this activity, leading to a more complete predictive model of the system for the user. We can apply generalization to situations in which the user wants to apply knowledge that helps achieve one particular goal to another situation where the goal is in some way similar. Generalizability can be seen as a form of consistency.

Generalization can occur within a single application or across a variety of applications. For example, in a graphical drawing package that draws a circle as a constrained form of ellipse, we would want the user to generalize that a square can be drawn as a constrained rectangle. A good example of generalizability across a variety of applications can be seen in multi-windowing systems that attempt to provide cut/paste/copy operations to all applications in the same way (with varying degrees of success). Generalizability within an application can be maximized by any conscientious designer. One of the main advantages of standards and programming style guides, which we will discuss in Sections 7.3 and 7.4, is that they increase generalizability across a wide variety of applications within the same environment.

Consistency

Consistency relates to the likeness in behavior arising from similar situations or similar task objectives. Consistency is probably the most widely mentioned principle in the literature on user interface design. 'Be consistent!' we are constantly urged. The user relies on a consistent interface. However, the difficulty of dealing with consistency is that it can take many forms. Consistency is not a single property of an interactive system that is either satisfied or not satisfied. Instead, consistency must be applied relative to something. Thus we have consistency in command naming, or consistency in command/argument invocation.

Another consequence of consistency having to be defined with respect to some other feature of the interaction is that many other principles can be 'reduced' to qualified instances of consistency. Hence, familiarity can be considered as consistency with respect to past real-world experience, and generalizability as consistency with respect to experience with the same system or set of applications on the same platform. Because of this pervasive quality of consistency, it might be argued that consistency should be a separate category of usability principles, on the same level as learnability, flexibility and robustness. Rather than do that, we will discuss different ways in which consistency can be manifested.

Consistency can be expressed in terms of the form of input expressions or output responses with respect to the meaning of actions in some conceptual model of the system. For example, before the introduction of explicit arrow keys, some word processors used the relative position of keys on the keyboard to indicate directionality for operations (for example, to move one character to the left, right, up or down). The conceptual model for display-based editing is a two-dimensional plane, so the user would think of certain classes of operations in terms of movements up, down, left or right in the plane of the display. Operations that required directional information, such as moving within the text or deleting some unit of text, could be articulated by using some set of keys on the keyboard that form a pattern consistent with up, down, left and right (for example, the keys e, x, s and d, respectively). For output responses, a good example of consistency can be found in a warnings system for an aircraft. Warnings to the pilot are classified into three categories, depending on whether the situation with the aircraft requires immediate recovery action, eventual but not immediate action, or no action at all (advisory) on the part of the crew. These warnings are signalled to the crew by means of a centralized warnings panel in which the categories are consistently color coded (red for immediate, amber for eventual and green for advisory).

Grudin has argued that because of the relative nature of consistency it can be a dangerous principle to follow [160]. A good example he gives is the development and evolution of the standard typewriter keyboard. When keyboards for typewriters were first made, the designers laid out the keys in alphabetical order. Then it was discovered that such an arrangement of keys was both inefficient from the machine's perspective (adjacent typewriter keys pressed in succession caused jams in the mechanism, so the likelihood of this occurrence had to be designed out) and tiring for the typist (a touch-typist would not have equal stress distributed over all fingers). The resulting QWERTY and DVORAK keyboards have since been adopted to combat the problems of the 'consistent' keyboard layout.[1]

1 See Chapter 2 for a discussion of different keyboards.

Table 7.2 Summary of principles affecting flexibility

Principle	Definition	Related principles
Dialog initiative	Allowing the user freedom from artificial constraints on the input dialog imposed by the system	System/user pre-emptiveness
Multi-threading	Ability of the system to support user interaction pertaining to more than one task at a time	Concurrent vs. interleaving, multi-modality
Task migratability	The ability to pass control for the execution of a given task so that it becomes either internalized by the user or the system or shared between them	–
Substitutivity	Allowing equivalent values of input and output to be arbitrarily substituted for each other	Representation multiplicity, equal opportunity
Customizability	Modifiability of the user interface by the user or the system	Adaptivity, adaptability

7.2.2 Flexibility

Flexibility refers to the multiplicity of ways in which the end-user and the system exchange information. We identify several principles that contribute to the flexibility of interaction, and these are summarized in Table 7.2.

Dialog initiative

When considering the interaction between user and system as a dialog between partners (see Chapter 16), it is important to consider which partner has the initiative in the conversation. The system can initiate all dialog, in which case the user simply responds to requests for information. We call this type of dialog *system pre-emptive*. For example, a modal dialog box prohibits the user from interacting with the system in any way that does not direct input to the box. Alternatively, the user may be entirely free to initiate any action towards the system, in which case the dialog is *user pre-emptive*. The system may control the dialog to the extent that it prohibits the user from initiating any other desired communication concerning the current task or some other task the user would like to perform. From the user's perspective, a system-driven interaction hinders flexibility whereas a user-driven interaction favours it.

In general, we want to maximize the user's ability to pre-empt the system and minimize the system's ability to pre-empt the user. Although a system pre-emptive dialog is not desirable in general, some situations may require it. In a cooperative editor (in which two people edit a document at the same time) it would be impolite for you to erase a paragraph of text that your partner is currently editing. For safety reasons, it may be necessary to prohibit the user from the 'freedom' to do potentially serious damage. A pilot about to land an aircraft in which the flaps have asymmetrically failed in their extended position[2] should not be allowed to abort the landing, as this failure will almost certainly result in a catastrophic accident.

On the other hand, a completely user pre-emptive dialog allows the user to offer any input action at any time for maximum flexibility. This is not an entirely desirable situation, since it increases the likelihood that the user will lose track of the tasks that have been initiated and not yet completed. However, if the designers have a good understanding of the sets of tasks the user is likely to perform with a system and how those tasks are related, they can minimize the likelihood that the user will be prevented from initiating some task at a time when he wishes to do so.

Multi-threading

A thread of a dialog is a coherent subset of that dialog. In the user–system dialog, we can consider a thread to be that part of the dialog that relates to a given user task. *Multi-threading* of the user–system dialog allows for interaction to support more than one task at a time. *Concurrent* multi-threading allows simultaneous communication of information pertaining to separate tasks. *Interleaved* multi-threading permits a temporal overlap between separate tasks, but stipulates that at any given instant the dialog is restricted to a single task.

Multi-modality of a dialog is related to multi-threading. Coutaz has characterized two dimensions of multi-modal systems [80]. First, we can consider how the separate modalities (or channels of communication) are combined to form a single input or output expression. Multiple channels may be available, but any one expression may be restricted to just one channel (keyboard or audio, for example). As an example, to open a window the user can choose between a double click on an icon, a keyboard shortcut, or saying 'open window'. Alternatively, a single expression can be formed by a mixing of channels. Examples of such fused modality are error warnings, which usually contain a textual message accompanied by an audible beep. On the input side, we could consider chord sequences of input with a keyboard and mouse (pressing the shift key while a mouse button is pressed, or saying 'drop' as you drag a file over the trash icon). We can also characterize a multi-modality dialog depending on whether it allows concurrent or interleaved use of multiple modes.

A windowing system naturally supports a multi-threaded dialog that is interleaved amongst a number of overlapping tasks. Each window can represent a different task, for example text editing in one window, file management in another, a telephone directory in another and electronic mail in yet another. A multi-modal dialog can allow for concurrent multi-threading. A very simple example can occur in the

2 Flaps increase the surface area and curvature of the aircraft's wing, providing the extra lift necessary for, among other things, a smooth touchdown. An asymmetric failure results in extreme instability and the aircraft will not fly level.

windowing system with an audible bell. You are editing a program when a beep indicates that a new electronic mail message has arrived. Even though at the level of the system the audible beep has been interleaved with your requests from the keyboard to perform edits, the overlap between the editing task and the mail message from your perspective is simultaneous.

Task migratability

Task migratability concerns the transfer of control for execution of tasks between system and user. It should be possible for the user or system to pass the control of a task over to the other or promote the task from a completely internalized one to a shared and cooperative venture. Hence, a task that is internal to one can become internal to the other or shared between the two partners.

Spell-checking a paper is a good example of the need for task migratability. Equipped with a dictionary, you are perfectly able to check your spelling by reading through the entire paper and correcting mistakes as you spot them. This mundane task is perfectly suited to automation, as the computer can check words against its own list of acceptable spellings. It is not desirable, however, to leave this task completely to the discretion of the computer, as most computerized dictionaries do not handle proper names correctly, nor can they distinguish between correct and unintentional duplications of words. In those cases, the task is handed over to the user. The spell-check is best performed in such a cooperative way.

In safety-critical applications, task migratability can decrease the likelihood of an accident. For example, on the flight deck of an aircraft, there are so many control tasks that must be performed that a pilot would be overwhelmed if he had to perform them all. Therefore, mundane control of the aircraft's position within its flight envelope is greatly automated. However, in the event of an emergency, it must be possible to transfer flying controls easily and seamlessly from the system to the pilot.

Substitutivity

Substitutivity requires that equivalent values can be substituted for each other. For example, in considering the form of an input expression to determine the margin for a letter, you may want to enter the value in either inches or centimeters. You may also want to input the value explicitly (say 1.5 inches) or you may want to enter a calculation which produces the right input value (you know the width of the text is 6.5 inches and the width of the paper is 8.5 inches and you want the left margin to be twice as large as the right margin, so you enter $\frac{2}{3}$ (8.5 – 6.5) inches). This input substitutivity contributes towards flexibility by allowing the user to choose whichever form best suits the needs of the moment. By avoiding unnecessary calculations in the user's head, substitutivity can minimize user errors and cognitive effort.

We can also consider substitutivity with respect to output, or the system's rendering of state information. *Representation multiplicity* illustrates flexibility for state rendering. For example, the temperature of a physical object over a period of time can be presented as a digital thermometer if the actual numerical value is important or as a graph if it is only important to notice trends. It might even be desirable to make these representations simultaneously available to the user. Each representation provides a perspective on the internal state of the system. At a given time, the user is free to consider the representations that are most suitable for the current task.

Equal opportunity blurs the distinction between input and output at the interface. The user has the choice of what is input and what is output; in addition, output can be reused as input. Thimbleby describes this principle as, 'If you can see it, you can use it!' It is a common belief that input and output are separate. Many have stressed the significance of the link between input and output. Equal opportunity pushes that view to the extreme. For example, in spreadsheet programs, the user fills in some cells and the system automatically determines the values attributed to some other cells. Conversely, if the user enters values for those other cells, the system would compute the values for the first ones. In this example, it is not clear which cells are the inputs and which are the outputs. Furthermore, this distinction might not be clear or useful to the user. In a drawing package, the user may draw a line by direct manipulation and the system would compute the length of the line; or conversely, the user may specify the line coordinates and the system would draw the line. Both means of manipulating the line are equally important and must be made equally available. Note that equal opportunity implies that the system is not pre-emptive towards the user.

Customizability

Customizability is the modifiability of the user interface by the user or the system. From the system side, we are not concerned with modifications that would be attended to by a programmer actually changing the system and its interface during system maintenance. Rather, we are concerned with the automatic modification that the system would make based on its knowledge of the user. We distinguish between the user-initiated and system-initiated modification, referring to the former as *adaptability* and the latter as *adaptivity*.

Adaptability refers to the user's ability to adjust the form of input and output. This customization could be very limited, with the user only allowed to adjust the position of soft buttons on the screen or redefine command names. This type of modifiability, which is restricted to the surface of the interface, is referred to as lexical customization. The overall structure of the interaction is kept unchanged. The power given to the user can be increased by allowing the definition of macros to speed up the articulation of certain common tasks. In the extreme, the interface can provide the user with programming language capabilities, such as the UNIX shell or the script language Hypertalk in HyperCard. Thimbleby points out that in these cases it would be suitable to apply well-known principles of programming languages to the user's interface programming language.

Adaptivity is automatic customization of the user interface by the system. Decisions for adaptation can be based on user expertise or observed repetition of

Table 7.3 Summary of principles affecting robustness

Principle	Definition	Related principles
Observability	Ability of the user to evaluate the internal state of the system from its perceivable representation	Browsability, static/dynamic defaults, reachability, persistence, operation visibility
Recoverability	Ability of the user to take corrective action once an error has been recognized	Reachability, forward/backward recovery, commensurate effort
Responsiveness	How the user perceives the rate of communication with the system	Stability
Task conformance	The degree to which the system services support all of the tasks the user wishes to perform and in the way that the user understands them	Task completeness, task adequacy

certain task sequences. The distinction between adaptivity and adaptability is that the user plays an explicit role in adaptability, whereas his role in an adaptive interface is more implicit. A system can be trained to recognize the behavior of an expert or novice and accordingly adjust its dialog control or help system automatically to match the needs of the current user. This is in contrast with a system that would require the user to classify himself as novice or expert at the beginning of a session. We discuss adaptive systems further in the context of user support in Chapter 11. Automatic macro construction is a form of programming by example, combining adaptability with adaptivity in a simple and useful way. Repetitive tasks can be detected by observing user behavior and macros can be automatically (or with user consent) constructed from this observation to perform repetitive tasks automatically.

7.2.3 Robustness

In a work or task domain, a user is engaged with a computer in order to achieve some set of goals. The robustness of that interaction covers features that support the successful achievement and assessment of the goals. Here, we describe principles that support robustness. A summary of these principles is presented in Table 7.3.

Observability

Observability allows the user to evaluate the internal state of the system by means of its perceivable representation at the interface. As we described in Chapter 3, evaluation allows the user to compare the current observed state with his intention within the task–action plan, possibly leading to a plan revision. Observability can be discussed through five other principles: browsability, defaults, reachability, persistence and operation visibility. Operation visibility was covered in Section 7.2.1 in relation to predictability. The remaining four are discussed next.

Browsability allows the user to explore the current internal state of the system via the limited view provided at the interface. Usually the complexity of the domain does not allow the interface to show all of the relevant domain concepts at once. Indeed, this is one reason why the notion of task is used, in order to constrain the domain information needed at one time to a subset connected with the user's current activity. While you may not be able to view an entire document's contents, you may be able to see all of an outline view of the document, if you are only interested in its overall structure. Even with a restriction of concepts relevant to the current task, it is probable that all of the information a user needs to continue work on that task is not immediately perceivable. Or perhaps the user is engaged in a multi-threaded dialog covering several tasks. There needs to be a way for the user to investigate, or browse, the internal state. This browsing itself should not have any side-effects on that state; that is, the browsing commands should be passive with respect to the domain-specific parts of the internal state.

The availability of *defaults* can assist the user by passive recall (for example, a suggested response to a question can be recognized as correct instead of recalled). It also reduces the number of physical actions necessary to input a value. Thus, providing default values is a kind of error prevention mechanism. There are two kinds of default values: static and dynamic. Static defaults do not evolve with the session. They are either defined within the system or acquired at initialization. On the other hand, dynamic defaults evolve during the session. They are computed by the system from previous user inputs; the system is then adapting default values.

Reachability refers to the possibility of navigation through the observable system states. There are various levels of reachability that can be given precise mathematical definitions (see Chapter 17), but the main notion is whether the user can navigate from any given state to any other state. Reachability in an interactive system affects the recoverability of the system, as we will discuss later. In addition, different levels of reachability can reflect the amount of flexibility in the system as well, though we did not make that explicit in the discussion on flexibility.

Persistence deals with the duration of the effect of a communication act and the ability of the user to make use of that effect. The effect of vocal communication does not persist except in the memory of the receiver. Visual communication, on the other hand, can remain as an object which the user can subsequently manipulate long after the act of presentation. If you are informed of a new email message by a beep at your terminal, you may know at that moment and for a short while later that you have received a new message. If you do not attend to that message immediately, you may forget about it. If, however, some persistent visual information informs you of the incoming message (say, the flag goes up on your electronic mailbox), then that will serve as a reminder that an unread message remains long after its initial receipt.[3]

3 Chapter 19 discusses notification mechanisms for email in more detail.

Recoverability

Users make mistakes from which they want to recover. Recoverability is the ability to reach a desired goal after recognition of some error in a previous interaction. There are two directions in which recovery can occur, forward or backward. *Forward error recovery* involves the acceptance of the current state and negotiation from that state towards the desired state. Forward error recovery may be the only possibility for recovery if the effects of interaction are not revocable (for example, in building a house of cards, you might sneeze whilst placing a card on the seventh level, but you cannot undo the effect of your misfortune except by rebuilding). *Backward error recovery* is an attempt to undo the effects of previous interaction in order to return to a prior state before proceeding. In a text editor, a mistyped keystroke might wipe out a large section of text which you would want to retrieve by an equally simple undo button.

Recovery can be initiated by the system or by the user. When performed by the system, recoverability is connected to the notions of fault tolerance, safety, reliability and dependability, all topics covered in software engineering. However, in software engineering this recoverability is considered only with respect to system functionality; it is not tied to user intent. When recovery is initiated by the user, it is important that it determines the intent of the user's recovery actions; that is, whether he desires forward (negotiation) or backward (using undo/redo actions) corrective action.

Recoverability is linked to reachability because we want to avoid blocking the user from getting to a desired state from some other undesired state (going down a blind alley).

In addition to providing the ability to recover, the procedure for recovery should reflect the work being done (or undone, as the case may be). The principle of *commensurate effort* states that if it is difficult to undo a given effect on the state, then it should have been difficult to do in the first place. Conversely, easily undone actions should be easily doable. For example, if it is difficult to recover files which have been deleted in an operating system, then it should be difficult to remove them, or at least it should require more effort by the user to delete the file than to, say, rename it.

Responsiveness

Responsiveness measures the rate of communication between the system and the user. Response time is generally defined as the duration of time needed by the system to express state changes to the user. In general, short durations and instantaneous response times are desirable. Instantaneous means that the user perceives system reactions as immediate. But even in situations in which an instantaneous response cannot be obtained, there must be some indication to the user that the system has received the request for action and is working on a response.

As significant as absolute response time is response time *stability*. Response time stability covers the invariance of the duration for identical or similar computational resources. For example, pull-down menus are expected to pop up instantaneously as soon as a mouse button is pressed. Variations in response time will impede anticipation exploited by motor skill.

Task conformance

Since the purpose of an interactive system is to allow a user to perform various tasks in achieving certain goals within a specific application domain, we can ask whether the system supports all of the tasks of interest and whether it supports these as the user wants. *Task completeness* addresses the coverage issue and *task adequacy* addresses the user's understanding of the tasks.

It is not sufficient that the computer system fully implements some set of computational services that were identified at early specification stages. It is essential that the system allows the user to achieve any of the desired tasks in a particular work domain as identified by a task analysis that precedes system specification (see Chapter 15 for a more complete discussion of task analysis techniques). Task completeness refers to the level to which the system services can be mapped onto all of the user tasks. However, it is quite possible that the provision of a new computer-based tool will suggest to a user some tasks that were not even conceivable before the tool. Therefore, it is also desirable that the system services be suitably general so that the user can define new tasks.

Discussion of task conformance has its roots in an attempt to understand the success of direct manipulation interfaces. We can view the direct manipulation interface as a separate world from that inside the system. Task completeness covers only one part of the conformance. This separate world is understood and operated upon by the user. With the intuition of the Hutchins, Hollan and Norman model-world metaphor discussed in Chapter 4, we require that the task, as represented by the world of the interface, matches the task as understood by the user and supported by the system. If the model-world metaphor satisfies the principle of task adequacy, then the user will be directly on his task plan, minimizing the effort required in the articulation and observation translations discussed in the interaction framework of Chapter 3.

Worked exercise

Look at some of the principles outlined in this section, and use one or two to provide a usability specification (see Chapter 6, Section 6.3) for an electronic meetings diary or calendar. First identify some of the tasks that would be performed by a user trying to keep track of future meetings, and then complete the usability specification assuming that the electronic system will be replacing a paper-based system. What assumptions do you have to make about the user and the electronic diary in order to create a reasonable usability specification?

Answer This exercise could be easily extended to a small project which would involve the design of such an electronic diary or calendar. The purpose of this smaller usability engineering exercise is to show how usability goals can be formulated early on to drive the design activity. We will select two of the usability principles from this chapter, which will serve as attributes for separate usability specifications.

In the first example, we will consider the interaction principle of guessability, which concerns how easy it is for new users to perform tasks initially. The measuring concept will be how long it takes a new user, without any instruction on the new system, to enter his first appointment in the diary. A sample usability specification is given below.

Attribute:	Guessability
Measuring concept:	Ease of first use of system without training
Measuring method:	Time to create first entry in diary
Now level:	30 seconds on paper-based system
Worst case:	1 minute
Planned level:	45 seconds
Best case:	30 seconds (equivalent to now)

The values in this usability specification might seem a little surprising at first, since we are saying that the best case is only equivalent to the currently achievable now level. The point in this example is that the new system is replacing a very familiar paper and pencil system which requires very little training. The objective of this system is not so much to improve guessability but to preserve it. Earlier, we discussed that the worst case level should not usually be worse than the now level, but we are hoping for this product to improve overall functionality of the system. The user will be able to do more things with the electronic diary than he could with the conventional system. As a result, we worry less about improving its guessability. Perhaps we could have been more ambitious in setting the best case value by considering the potential for voice input or other exotic input techniques that would make entry faster than writing.

As another example, we want to support the task migratability of the system. A frequent sort of task for a diary is to schedule weekly meetings. The conventional system would require the user to make an explicit entry for the meeting each week – the task of the scheduling is the responsibility of the user. In the new system, we want to allow the user to push the responsibility of scheduling over to the system, so that the user need only indicate the desire to have a meeting scheduled for a certain time each week and the system will take care of entering the meeting at all of the appropriate times. The task of scheduling has thus migrated over to the system. The usability specification for this example follows.

Attribute:	Task migratability
Measuring concept:	Scheduling a weekly meeting
Measuring method:	Time it takes to enter a weekly meeting appointment
Now level:	(Time to schedule one appointment) × (Number of weeks)
Worst case:	Time to schedule two appointments
Planned level:	1.5 × (Time to schedule one appointment)
Best case:	Time to schedule one appointment

In this specification, we have indicated that the now level is equivalent to the time it takes to schedule each appointment separately. The worst, planned and best case levels are all targeted at some proportion of the time it takes to schedule just a single appointment – a dramatic improvement. The difference between the worst, planned and best case levels is the amount of overhead it will take to indicate that a single appointment is to be considered an example to be repeated at the weekly level.

What are the assumptions we have to make in order to arrive at such a usability specification? One of the problems with usability specifications, discussed earlier, is that they sometimes require quite specific information about the design. For example, had we set one of our measuring methods to count keystrokes or mouse clicks, we would have had to start making assumptions about the method of interaction that the system would allow. Had we tried to set a usability specification concerning the browsing of the diary, we would have had to start making assumptions about the layout of the calendar (monthly, weekly, daily) in order to make our estimates specific enough to measure. In the examples we have provided above, we have tried to stay as abstract as possible, so that the usability specifications could be of use as early in the design life cycle as possible. A consequence of this abstractness, particularly evident in the second example, is that we run the risk in the usability specification of setting goals that may be completely unrealistic, though well intentioned. If the usability specification were to be used as a contract with the customer, such speculation could spell real trouble for the designer.

7.3 STANDARDS

Standards for interactive system design are usually set by national or international bodies to ensure compliance with a set of design rules by a large community. Standards can apply specifically to either the hardware or the software used to build the interactive system. Smith [324] points out the differing characteristics between hardware and software, which affect the utility of design standards applied to them:

Underlying theory Standards for hardware are based on an understanding of physiology or ergonomics/human factors, the results of which are relatively well known, fixed and readily adaptable to design of the hardware. On the other hand, software standards are based on theories from psychology or cognitive science, which are less well formed, still evolving and not very easy to interpret in the language of software design. Consequently, standards for hardware can directly relate to a hardware specification and still reflect the underlying theory, whereas software standards would have to be more vaguely worded.

Change Hardware is more difficult and expensive to change than software, which is usually designed to be very flexible. Consequently, requirements changes for hardware do not occur as frequently as for software. Since standards are also relatively stable, they are more suitable for hardware than software.

Historically, for these reasons, a given standards institution, such as the British Standards Institution (BSI) or the International Organization for Standardization (ISO) or a national military agency, has had standards for hardware in place before any for software. For example, the UK Ministry of Defence has published an Interim Defence Standard 00–25 on *Human Factors for Designers of Equipment*, produced in 12 parts:

Part 1	Introduction
Part 2	Body Size
Part 3	Body Strength and Stamina
Part 4	Workplace Design
Part 5	Stresses and Hazards
Part 6	Vision and Lighting

Part 7	Visual Displays
Part 8	Auditory Information
Part 9	Voice Communication
Part 10	Controls
Part 11	Design for Maintainability
Part 12	Systems

Only the last of these is concerned with the software design process. The international standard ISO 9241, entitled *Ergonomic Requirements for Office Work with Visual Display Terminals (VDT)s*, has 17 parts. Seven of these are concerned with hardware issues – requirements for visual display, keyboard layout, workstation layout, environment, display with reflections, display colors and non-keyboard input devices. Seven parts are devoted to software issues – general dialog principles, menu dialogs, presentation of information, user guidance, command dialogs, direct manipulation dialogs and form-filling dialogs. However, standards covering software issues are now being produced, for example, the draft standard ISO 14915 covers software ergonomics for multimedia user interfaces.

Figure 7.1 provides examples of the language of standards for displays. Note the increasing generality and vagueness of the language as we progress from the hardware issues in a UK defence standard for pilot cockpit controls and instrumentation through a German standard for user interface design of display workstations to a US military standard for display contents.

11.3 *Arrangement of displays*
11.3.1 Vertical Grouping. The engine display parameters shall be arranged so that the primary or most important display for a particular engine and airplane (thrust, torque, RPM, etc.) be located at the top of the display group if a vertical grouping is provided. The next most important display parameter shall be positioned under the primary display progressing down the panel with the least important at the bottom.

(a) A typical example of a military standard

5.1 *Subdivision of the display area*
In consideration of a simple, fast and accurate visual acquisition, the display area shall be divided into different sub-areas.
Such a division should be:

- Input area
- Output area
- Area for operational indications (such as status and alarms)

(b) From German standard DIN 66 234 Part 3 (1984), adapted from Smith [324]

5.15.3.2.1 *Standardization*
The content of displays within a system shall be presented in a consistent manner.

(c) From US military standard MIL-STD-1472C, revised (1983), adapted from Smith [324]

Figure 7.1 Sample design standards for displays. Adapted from Smith [324]. Copyright © 1986 IEEE

One component of the ISO standard 9241, pertaining to usability specification, applies equally to both hardware and software design. In the beginning of that document, the following definition of usability is given:

Usability The effectiveness, efficiency and satisfaction with which specified users achieve specified goals in particular environments.

Effectiveness The accuracy and completeness with which specified users can achieve specified goals in particular environments.

Efficiency The resources expended in relation to the accuracy and completeness of goals achieved.

Satisfaction The comfort and acceptability of the work system to its users and other people affected by its use.

The importance of such a definition in the standard is as a means of describing explicit measurements for usability. Such metrics can support usability engineering, as we saw in Chapter 6.

The strength of a standard lies in its ability to force large communities to abide – the so-called authority we have referred to earlier. It should be noted that such authority does not necessarily follow from the publication of a standard by a national or international body. In fact, many standards applying to software design are put forth as suggestive measures, rather than obligatory. The authority of a standard (or a guideline, for that matter) can only be determined from its use in practice. Some software products become de facto standards long before any formal standards document is published (for example, the X windowing system).

There is a much longer history of standards in safety-critical domains, such as nuclear power plants or aircraft design, where the consequences of poor design outweigh the expense of principled design. It is only as the perceived costs of unusable software in less safety-critical domains have become less acceptable that there has been a greater effort in developing standards for promoting usability.

7.4 GUIDELINES

We have observed that the incompleteness of theories underlying the design of interactive software makes it difficult to produce authoritative and specific standards. As a result, the majority of design rules for interactive systems are suggestive and more general guidelines. Our concern in examining the wealth of available guidelines is in determining their applicability to the various stages of design. The more abstract the guideline, the more it resembles the principles that we outlined in Section 7.2, which would be most suited to requirements specification. The more specific the guideline, the more suited it is to detailed design. The guidelines can also be automated to some extent, providing a direct means for translating detailed design specifications into actual implementation. There are a vast amount of published guidelines for interactive system design (they are frequently referred to as guidelines for user interface design). We will present only a few examples here to demonstrate the content of guidelines in that vast literature.

Several books and technical reports contain huge catalogs of guidelines. A classic example was a very general list compiled by Smith and Mosier in 1986 at the Mitre Corporation and sponsored by the Electronic Systems Division of the US Air Force [325]. The basic categories of the Smith and Mosier guidelines are:

1. Data Entry
2. Data Display
3. Sequence Control
4. User Guidance
5. Data Transmission
6. Data Protection

Each of these categories is further broken down into more specific subcategories which contain the particular guidelines. Figure 7.2 provides an example of the information contained in the Smith and Mosier guidelines. A striking feature of this compendium of guidelines is the extensive cross-referencing within the catalog, and citation to published work that supports each guideline. The Mitre Corporation has taken advantage of this structure and implemented the Smith and Mosier guidelines on a hypertext system, which provides rapid traversal of the network of guidelines to investigate the cross-references and citations.

1. **Data Entry**

1.1 *Position Designation*

1.1–1 **Distinctive Cursor**
For position designation on an electronic display, provide a movable cursor with distinctive visual features (shape, blink, etc.).

Exception When position designation involves only selection among displayed alternatives, highlighting selected items might be used instead of a separately displayed cursor.

Comment When choosing a cursor shape, consider the general content of the display. For instance, an underscore cursor would be difficult to see on a display of underscored text, or on a graphical display containing many other lines.

Comment If the cursor is changed to denote different functions (e.g. to signal deletion rather than entry), then each different cursor should be distinguishable from the others.

Comment If multiple cursors are used on the same display (e.g. one for alphanumeric entry and one for line drawing), then each cursor should be distinguishable from the others.

Reference Whitfield, Ball and Bird, 1983

See also 1.1–17 Distinctive multiple cursors
4.0–9 Distinctive cursor

Figure 7.2 Sample guideline from Smith and Mosier [325], courtesy of The MITRE Corporation

Table 7.4 Comparison of dialog styles mentioned in guidelines

Smith and Mosier [325]	Mayhew [230]
Question and answer	Question and answer
Form filling	Fill-in forms
Menu selection	Menus
Function keys	Function keys
Command language	Command language
Query language	–
Natural language	Natural language
Graphic selection	Direct manipulation

A more recent, equally comprehensive catalog of general guidelines has been compiled by Mayhew [230]. Though this catalog is only in book form, and so limits the possibility of quick cross-referencing, this is one of the best sources for the experimental results which back the specific guidelines.

A major concern for all of the general guidelines is the subject of *dialog styles*, which in the context of these guidelines pertains to the means by which the user communicates input to the system, including how the system presents the communication device. Smith and Mosier identify eight different dialog styles and Mayhew identifies seven (see Table 7.4 for a comparison). The only real difference is the absence of query languages in Mayhew's list, but we can consider a query language as a special case of a command language. These interface styles have been described in more detail in Chapter 3.

Most guidelines are applicable for the implementation of any one of these dialog styles in isolation. It is also important to consider the possibility of mixing dialog styles in one application. In contrasting the action and language paradigms in Chapter 4, we concluded that it is not always the case that one paradigm wins over the other for all tasks in an application and, therefore, an application may want to mix the two paradigms. This equates to a mixing of dialog styles – a direct manipulation dialog being suitable for the action paradigm and a command language being suitable for the language paradigm. Mayhew provides guidelines and a technique for deciding how to mix dialog styles.

In moving from abstract guidelines to more specific and automated ones, it is necessary to introduce assumptions about the computer platform on which the interactive system is designed. So, for example, in Apple's *Human Interface Guidelines: the Apple Desktop Interface*, there is a clear distinction between the abstract guidelines (or principles), independent of the specific Macintosh hardware and software, and the concrete guidelines, which assume them. The abstract guidelines provide the so-called philosophy of programming that Apple would like designers to adopt in programming applications for the Macintosh. The more concrete guidelines are then seen as more concrete manifestations of that philosophy.

As an example, one abstract principle espoused in the Apple guidelines is *consistency*:

Effective applications are both consistent within themselves and consistent with one another.

We discussed consistency in Section 7.2 under the larger usability category of learnability, and the meaning in this context is similar. A more concrete directive that Apple provides is the 'noun–verb' ordering guideline: the user first selects an object (the noun) from the visible set on the Desktop and then selects an operation (the verb) to be applied to the object. For the sake of consistency, this ordering guideline is to be followed for all operation invocation involving the explicit and separate indication of an operation and the object or arguments of that operation.

Another less straightforward example from the Apple guidelines refers to user control:

The user, not the computer, initiates and controls all actions.

We considered issues of dialog initiative in Section 7.2 under the general usability category of flexibility. As we mentioned there, the issue of dialog initiative involves a trade-off between user freedom and system protection. In general, single-user computer systems operate in strict abidance of this guideline for user control; the user is allowed to initiate any dialog at all with the computer, whether or not it will have the intended result. Part of the success of direct manipulation interfaces lies in their ability to constrain user interaction to actions which are both syntactically correct (for example, preventing errors due to slips in typing) and will probably correspond to the intended user tasks.

Other popular graphical user interface (GUI) systems have published guidelines that describe how to adhere to abstract principles for usability in the narrower context of a specific programming environment. These guidelines are often referred to as *style guides* to reflect that they are not hard and fast rules, but suggested conventions for programming in that environment. Some examples are the OpenLook and the Open Software Foundation (OSF) Motif graphical user interfaces, both of which have published style guides [337, 275]. Programming in the style of these GUIs involves the use of toolkits which provide high-level widgets, as we have mentioned earlier in this book and will discuss in more detail in Chapter 8. More importantly, each of these GUIs has its own *look and feel*, which describes their expected behavior. The style guides are intended to help a programmer capture the elements of the look and feel of a GUI in her own programming. Therefore, style guides for the look and feel of a GUI promote the consistency within and between applications on the same computer platform.

We discussed menus in Chapter 3 as one of the major elements of the WIMP interface. As one example of a guideline for the design of menus, the OpenLook style guide suggests the following for grouping items in the same menu:

Use white space between long groups of controls on menus or in short groups when screen real estate is not an issue.

The justification for such a guideline is that the more options (or controls, as the term is used in the quoted guideline) on a menu, the longer it will take a user to locate and point to a desired item. As we discussed in Chapter 1, humans chunk related information in the learning process and this can be used to increase the efficiency of searching. Grouping of related items in a menu can supplement this chunking procedure. But be warned! Remember the scenario described in the Introduction to this book, in which we fell victim to closely grouped menu items which had drastically different effects in our word processor. Saving and deleting files might be considered logically similar since they both deal with operations on the file level. But simple slips made in pointing (which are all too easy with trackball devices) can change an intended save operation into an unintended and dangerous delete.

Worked exercise *Look up and report back guidelines for the use of color. Be able to state the empirical psychological evidence that supports the guidelines. Do the guidelines conflict with any other known guidelines? Which principles of interaction do they support?*

Answer There are many examples of guidelines for the use of color in the literature. Here are three good sources:

- C. Marlin Brown, *Human–Computer Interface Design Guidelines*, Ablex, 1988.
- Deborah J. Mayhew, *Principles and Guidelines in Software User Interface Design*, Prentice Hall, 1992.
- Sun Microsystems, Inc., *OpenLook Graphical User Interface Application Style Guidelines*, Addison-Wesley, 1990.

Taking an example from Mayhew, we have the following design guideline for the use of color as an informational cue for the user (for example, to inform the user that a string of text is a warning or error message):

Do not use color without some other redundant cue.

Mayhew provides three reasons which empirically support this guideline:

1. Color may not be available on all machines on which the system is to be implemented. Therefore, if use of color is the only means to convey some important information to the user, then that information will be lost in a monochrome (no color) system. Redundant color coding will allow for portability across different computing platforms.
2. Empirical evidence shows that 8% of the (general) male population and 0.4% of the female population has some color deficiency, so they cannot accurately recognize or distinguish between various colors. Again, if color is the only means for conveying some information, this significant portion of the user population will be slighted.
3. It has been shown that redundant color coding enhances user performance

This guideline supports several of the principles discussed in this chapter:

Substitutivity The system is able to substitute color-coded information and other means (for example, text, sound) to represent some important information. We could turn the argument around and suggest that the user be able to provide color input (by selecting from a palette menu) or other forms of input to provide relevant information to the system.

Observability This principle is all about the system being able to provide the user with enough information about its internal state to assist his task. Relying strictly on color-coded information, as pointed out above, could reduce the observability of a system for some users.

Synthesis If a change in color is used to indicate the changing status of some system entity (perhaps a change in temperature above a threshold value is signalled by an icon becoming red), those who cannot detect the change in color would be deprived of this information. Synthesis is about supporting the user's ability to detect such significant changes, especially when they are a result of previous user actions.

There is no evidence of existing guidelines that this particular guideline for color violates.

Another example of a color guideline (found in all three of the above references) is the demand to consider cultural information in the selection of particular colors. For example, Mayhew states that western cultures tend to interpret green to mean go or safe; red to mean stop, on, hot or emergency; and blue to mean cold or off. Using color to suggest these kinds of meanings is in support of the familiarity principle within learnability. However, in other cultures different meanings may be associated with these colors, as we saw in Chapter 3, and consistent use of color (another guideline) might lead to confusion. Hence, strict adherence to this guideline would suggest a violation of the consistency of color application guideline. However, if consistency is applied relative to the meaning of the color (as opposed to its actual color), this guideline would not have to conflict.

7.5 GOLDEN RULES AND HEURISTICS

So far we have considered a range of abstract principles and detailed guidelines, which can be used to help designers produce more usable systems. But all of these rules require a certain amount of commitment on the part of the designer, either to track down appropriate guidelines or to interpret principles. Is there a simpler way?

A number of advocates of user-centered design have presented sets of 'golden rules' or heuristics. While these are inevitably 'broad-brush' design rules, which may not be always be applicable to every situation, they do provide a useful checklist or summary of the essence of design advice. It is clear that any designer following even these simple rules will produce a better system than one who ignores them.

There are many sets of heuristics, but the most well used are Nielsen's ten heuristics, Shneiderman's eight golden rules and Norman's seven principles. Nielsen's heuristics are intended to be used in evaluation and will therefore be discussed in Chapter 9. We will consider the other two sets here.

7.5.1 Shneiderman's Eight Golden Rules of Interface Design

Shneiderman's eight golden rules provide a convenient and succinct summary of the key principles of interface design. They are intended to be used during design but can also be applied, like Nielsen's heuristics, to the evaluation of systems. Notice how they relate to the abstract principles discussed earlier.

1. *Strive for consistency* in action sequences, layout, terminology, command use and so on.
2. *Enable frequent users to use shortcuts*, such as abbreviations, special key sequences and macros, to perform regular, familiar actions more quickly.
3. *Offer informative feedback* for every user action, at a level appropriate to the magnitude of the action.
4. *Design dialogs to yield closure* so that the user knows when they have completed a task.
5. *Offer error prevention and simple error handling* so that, ideally, users are prevented from making mistakes and, if they do, they are offered clear and informative instructions to enable them to recover.
6. *Permit easy reversal of actions* in order to relieve anxiety and encourage exploration, since the user knows that he can always return to the previous state.
7. *Support internal locus of control* so that the user is in control of the system, which responds to his actions.
8. *Reduce short-term memory load* by keeping displays simple, consolidating multiple page displays and providing time for learning action sequences.

These rules provide a useful shorthand for the more detailed sets of principles described earlier. Like those principles, they are not applicable to every eventuality and need to be interpreted for each new situation. However, they are broadly useful and their application will only help most design projects.

7.5.2 Norman's Seven Principles for Transforming Difficult Tasks into Simple Ones

In Chapter 3 we discussed Norman's execution–evaluation cycle, in which he elaborates the seven stages of action. Later, in his classic book *The Design of Everyday Things*, he summarizes user-centered design using the following seven principles:

1. *Use both knowledge in the world and knowledge in the head*. People work better when the knowledge they need to do a task is available externally – either explicitly or through the constraints imposed by the environment. But experts also need to be able to internalize regular tasks to increase their efficiency. So systems should provide the necessary knowledge within the environment and their operation should be transparent to support the user in building an appropriate mental model of what is going on.
2. *Simplify the structure of tasks*. Tasks need to be simple in order to avoid complex problem solving and excessive memory load. There are a number of ways to simplify the structure of tasks. One is to provide mental aids to help the user keep track of stages in a more complex task. Another is to use technology to provide the user with more information about the task and better feedback. A third approach is to automate the task or part of it, as long as this does not detract from the user's experience. The final approach to simplification is to change the nature

of the task so that it becomes something more simple. In all of this, it is important not to take control away from the user.

3. *Make things visible*: bridge the gulfs of execution and evaluation. The interface should make clear what the system can do and how this is achieved, and should enable the user to see clearly the effect of their actions on the system.
4. *Get the mappings right*. User intentions should map clearly onto system controls. User actions should map clearly onto system events. So it should be clear what does what and by how much. Controls, sliders and dials should reflect the task – so a small movement has a small effect and a large movement a large effect.
5. *Exploit the power of constraints*, both natural and artificial. Constraints are things in the world that make it impossible to do anything but the correct action in the correct way. A simple example is a jigsaw puzzle, where the pieces only fit together in one way. Here the physical constraints of the design guide the user to complete the task.
6. *Design for error*. To err is human, so anticipate the errors the user could make and design recovery into the system.
7. *When all else fails, standardize*. If there are no natural mappings then arbitrary mappings should be standardized so that users only have to learn them once. It is this standardization principle that enables drivers to get into a new car and drive it with very little difficulty – key controls are standardized. Occasionally one might switch on the indicator lights instead of the windscreen wipers, but the critical controls (accelerator, brake, clutch, steering) are always the same.

Norman's seven principles provide a useful summary of his user-centered design philosophy but the reader is encouraged to read the complete text of *The Design of Everyday Things* to gain the full picture.

7.6 HCI PATTERNS

As we observed in Chapter 4, one way to approach design is to learn from examples that have proven to be successful in the past: to reuse the knowledge of what made a system – or paradigm – successful. Patterns are an approach to capturing and reusing this knowledge – of abstracting the essential details of successful design so that these can be applied again and again in new situations.

Patterns originated in architecture, where they have been used successfully, and they are also used widely in software development to capture solutions to common programming problems. More recently they have been used in interface and web design.

A pattern is an invariant solution to a recurrent problem within a specific context. Patterns address the problems that designers face by providing a 'solution statement'. This is best illustrated by example. Alexander, who initiated the pattern concept, proposes a pattern for house building called 'Light on Two Sides of Every Room'. The problem being addressed here is that

> When they have a choice, people will always gravitate to those rooms which have light on two sides, and leave the rooms which are lit only from one side unused and empty.

The proposed solution is to provide natural light from two sides of every room:

> Locate each room so that it has outdoor space outside it on at least two sides, and then place windows in these outdoor walls so that natural light falls into every room from more than one direction [9a, pattern 159].

Note that the solution says nothing about where these windows should be located or at what angle they should be to each other. A room with windows on opposite walls, or at right angles, or with a window and a skylight would all fulfill the pattern. Patterns capture only the invariant properties of good design – the common elements that hold between all instances of the solution. The specific implementation of the pattern will depend on the circumstance and the designer's creativity.

There are many examples of HCI patterns, and the interested reader is referred to pattern collections and languages such as [345, 37, 356] and the Pattern Gallery, which illustrates some of the various forms used in HCI patterns [132]. A well-known example, 'go back to a safe place', adapted from Tidwell's Common Ground collection, is given as an illustration (Figure 7.3). This is quite a low-level interface pattern, but patterns can also address high-level issues such as organizational structures or cooperative groups. As you can see, the pattern states the problem and the solution but also includes a rationale, explaining where the pattern has come from and in what context it applies, and examples to illustrate the pattern.

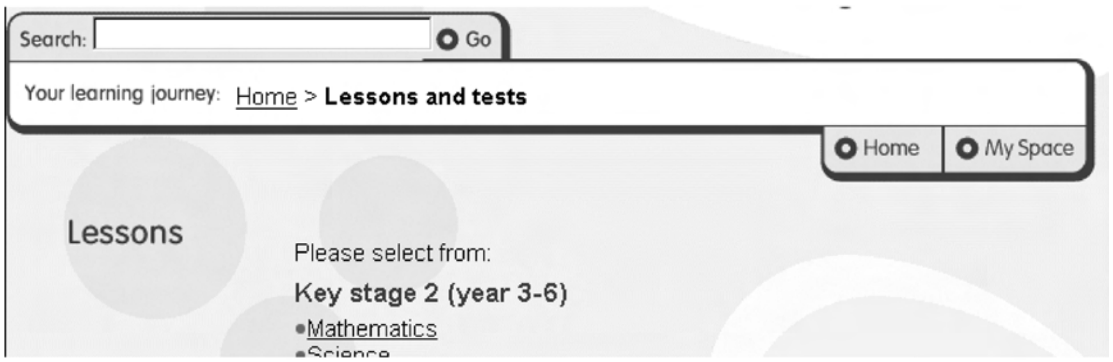
The pattern also has references to other patterns, indicating both the context in which it can be applied (the top references) and the patterns that may be needed to complete it (the bottom references). This connects the patterns together into a *language*. Patterns in isolation have limited use, but by traversing the hierarchy, through these references, the user is assisted in generating a complete design.

Patterns and pattern languages are characterized by a number of features, which, taken as a whole, distinguish them from other design rules:

- They capture design practice and embody knowledge about successful solutions: they come from practice rather than psychological theory.
- They capture the essential common properties of good design: they do not tell the designer *how* to do something but what needs to be done and why.
- They represent design knowledge at varying levels, ranging from social and organizational issues through conceptual design to detailed widget design.
- They are not neutral but embody values within their rationale. Alexander's language clearly expresses his values about architecture. HCI patterns can express values about what is humane in interface design.
- The concept of a pattern language is generative and can therefore assist in the development of complete designs.
- They are generally intuitive and readable and can therefore be used for communication between all stakeholders.

Patterns are a relatively recent addition to HCI representations, in which there are still many research issues to resolve. For instance, it is not clear how patterns can best be identified or how languages should be structured to reflect the temporal concerns of interaction. However, the recent publication of a complete pattern language for web design [356], aimed at commercial designers, may mark a turning point and see a more widespread adoption of the approach in interface design.

... NAVIGABLE SPACES or STEP BY STEP and a CONTROL PANEL are in place, which require the user to be able to move through the steps page by page.



From (www.learn.co.uk). An example of a toolbar that provides the option to return home and keeps track of the learning journey, providing the option to link back to a safe place.

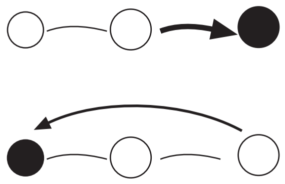
❖ ❖ ❖

It is easy to get lost in the tangle of links in a website.

People don't use the web like a TV or magazine. They use the web to find what they are looking for and then stop. They may select a wrong link and not be able to find their way back to something relevant. They do not always keep track of where they've been. They may forget where they have been if they are interrupted when using the site.

You are more likely to explore a website if you are sure that you can easily get out of an undesired state or space; that assurance engenders a feeling of security. Backtracking out of a long navigation path can be very tedious.

Therefore:
Always include a way back to a place that acts as a 'vantage point' from where you can reorientate yourself.



This pattern can be used in conjunction with GO BACK ONE STEP and CONTINUE TO NEXT STEP.

Figure 7.3 An example pattern 'go back to a safe place' adapted from Tidwell's Common Ground collection. Courtesy of Jenifer Tidwell

7.7 SUMMARY

We have seen how design rules can be used to provide direction for the design process, although the more general and interesting the design rule is for promoting usability, the further away it is from the actual language of design.

We have considered abstract principles, standards and guidelines, golden rules and heuristics, and patterns, and have looked at examples of each. The most abstract design rules are principles, which represent generic knowledge about good design practice. Standards and guidelines are more specific. Standards have the highest authority, being set by national or international bodies to ensure compliance by a large community. Guidelines are less authoritative but offer specific contextual advice, which can inform detailed design. Heuristics and 'golden rules' are succinct collections of design principles and advice that are easily assimilated by any designer. Patterns capture design practice and attempt to provide a generative structure to support the design process.

EXERCISES

7.1 What was the problem with the synthesis example comparing a command language interface with a visual interface? Can you suggest a fix to make a visual interface really immediately honest?

7.2 It has been suggested in this chapter that consistency could be considered a major category of interactive principles, on the same level as learnability, flexibility and robustness. If this was the case, which principles discussed in this chapter would appear in support of consistency?

7.3 Find as much information as you can on ISO standards that relate to usability. (**Hint:** Many standards are discussed in terms of ergonomics.) How many different standards and draft standards can you find?

7.4 Can you think of any instances in which the 'noun–verb' guideline for operations, as suggested in the Apple human interface guidelines for the Desktop Interface, would be violated? Suggest other abstract guidelines or principles besides consistency which support your example. (**Hint:** Think about moving files around on the Desktop.)

7.5 Can you think of any instances in which the user control guideline suggested by Apple is not followed? (**Hint:** Think about the use of dialog boxes.)

7.6 Find a book on guidelines. List the guidelines that are provided and classify them in terms of the activity in the software life cycle to which they would most likely apply.

7.7 (a) Distinguish between principles, guidelines and standards, using examples of each to illustrate.
(b) Why is context important in selecting and applying guidelines and principles for interface design? Illustrate your answer with examples.

7.8 (a) Why are there few effective HCI standards?
(b) How do 'golden rules' and heuristics help interface designers take account of cognitive psychology? Illustrate your answer with examples.

7.9 Using the web design pattern language in *The Design of Sites* [356] produce a design for an e-commerce site for a small retail business. How well does the language support the design process?

RECOMMENDED READING

H. Thimbleby, Design of interactive systems. In J. A. McDermid, editor, *The Software Engineer's Reference Book*, Chapter 57, Butterworth–Heinemann, 1992.
Thimbleby provides a very insightful list of general principles which apply to interactive systems. Some of the principles we have described in this chapter come from Thimbleby's work, though we have concentrated more on providing an overall organizational framework for the principles.

T. Stewart, Ergonomics user interface standards: are they more trouble than they are worth?, *Ergonomics*, Vol. 43, No. 7, pp. 1030–44, July 2000.
A review of the development of user interface standards, which considers the history of the endeavour, its successes and failures.

D. J. Mayhew, *Principles and Guidelines in Software User Interface Design*, Prentice Hall, 1992.
A comprehensive catalog of general interactive system guidelines which provides much of the experimental evidence to support specific guidelines.

B. Shneiderman, *Designing the User Interface*, 3rd edition, Addison-Wesley, 1998.
The source of Eight Golden Rules.

D. Norman, *The Design of Everyday Things*, MIT Press, 1998.
Classic exposition of design, including the seven principles.

D. Van Duyne, J. Landay and J. Hong, *The Design of Sites: Patterns, Principles and Processes for Crafting a Customer-centred Web Experience*, Addison-Wesley, 2003.
The first substantial and widely applicable HCI complete pattern language (for web design) presented accessibly for use in design teams.

The HCI Service, *HCI Tools & Methods Handbook*, 1991.
This booklet was produced under the United Kingdom Department of Trade and Industry (DTI) *Usability Now!* program. It contains a list of books on guidelines as well as a summary of the available user-centered design techniques as they can be applied to the software life cycle. The booklet and other information about the program is available from the HCI Service, PO Box 31, Loughborough, Leicestershire LE11 1QU, United Kingdom.

9 Evaluation techniques

OVERVIEW

- Evaluation tests the usability, functionality and acceptability of an interactive system.
- Evaluation may take place:
  - in the laboratory
  - in the field.
- Some approaches are based on expert evaluation:
  - analytic methods
  - review methods
  - model-based methods.
- Some approaches involve users:
  - experimental methods
  - observational methods
  - query methods.
- An evaluation method must be chosen carefully and must be suitable for the job.

9.1 WHAT IS EVALUATION?

In previous chapters we have discussed a design process to support the design of usable interactive systems. However, even if such a process is used, we still need to assess our designs and test our systems to ensure that they actually behave as we expect and meet user requirements. This is the role of evaluation.

Evaluation should not be thought of as a single phase in the design process (still less as an activity tacked on the end of the process if time permits). Ideally, evaluation should occur throughout the design life cycle, with the results of the evaluation feeding back into modifications to the design. Clearly, it is not usually possible to perform extensive experimental testing continuously throughout the design, but analytic and informal techniques can and should be used. In this respect, there is a close link between evaluation and the principles and prototyping techniques we have already discussed – such techniques help to ensure that the design is assessed continually. This has the advantage that problems can be ironed out before considerable effort and resources have been expended on the implementation itself: it is much easier to change a design in the early stages of development than in the later stages. We can make a broad distinction between evaluation by the designer or a usability expert, without direct involvement by users, and evaluation that studies actual use of the system. The former is particularly useful for assessing early designs and prototypes; the latter normally requires a working prototype or implementation. However, this is a broad distinction and, in practice, the user may be involved in assessing early design ideas (for example, through focus groups), and expert-based analysis can be performed on completed systems, as a cheap and quick usability assessment. We will consider evaluation techniques under two broad headings: expert analysis and user participation.

Before looking at specific techniques, however, we will consider why we do evaluation and what we are trying to achieve.

9.2 GOALS OF EVALUATION

Evaluation has three main goals: to assess the extent and accessibility of the system's functionality, to assess users' experience of the interaction, and to identify any specific problems with the system.

The system's functionality is important in that it must accord with the user's requirements. In other words, the design of the system should enable users to perform their intended tasks more easily. This includes not only making the appropriate functionality available within the system, but making it clearly reachable by the user in terms of the actions that the user needs to take to perform the task. It also involves matching the use of the system to the user's expectations of the task. For example, if a filing clerk is used to retrieving a customer's file by the postal address, the same capability (at least) should be provided in the computerized file system. Evaluation at this level may also include measuring the user's performance with the system, to assess the effectiveness of the system in supporting the task.

In addition to evaluating the system design in terms of its functional capabilities, it is important to assess the user's experience of the interaction and its impact upon him. This includes considering aspects such as how easy the system is to learn, its usability and the user's satisfaction with it. It may also include his enjoyment and emotional response, particularly in the case of systems that are aimed at leisure or entertainment. It is important to identify areas of the design that overload the user in some way, perhaps by requiring an excessive amount of information to be remembered, for example. A fuller classification of principles that can be used as evaluation criteria is provided in Chapter 7. Much evaluation is aimed at measuring features such as these.

The final goal of evaluation is to identify specific problems with the design. These may be aspects of the design which, when used in their intended context, cause unexpected results, or confusion amongst users. This is, of course, related to both the functionality and usability of the design (depending on the cause of the problem). However, it is specifically concerned with identifying trouble-spots which can then be rectified.

9.3 EVALUATION THROUGH EXPERT ANALYSIS

As we have noted, evaluation should occur throughout the design process. In particular, the first evaluation of a system should ideally be performed before any implementation work has started. If the design itself can be evaluated, expensive mistakes can be avoided, since the design can be altered prior to any major resource commitments. Typically, the later in the design process that an error is discovered, the more costly it is to put right and, therefore, the less likely it is to be rectified. However, it can be expensive to carry out user testing at regular intervals during the design process, and it can be difficult to get an accurate assessment of the experience of interaction from incomplete designs and prototypes. Consequently, a number of methods have been proposed to evaluate interactive systems through expert analysis. These depend upon the designer, or a human factors expert, taking the design and assessing the impact that it will have upon a typical user. The basic intention is to identify any areas that are likely to cause difficulties because they violate known cognitive principles, or ignore accepted empirical results. These methods can be used at any stage in the development process from a design specification, through storyboards and prototypes, to full implementations, making them flexible evaluation approaches. They are also relatively cheap, since they do not require user involvement. However, they do not assess actual use of the system, only whether or not a system upholds accepted usability principles.

We will consider four approaches to expert analysis: cognitive walkthrough, heuristic evaluation, the use of models and use of previous work.

9.3.1 Cognitive walkthrough

Cognitive walkthrough was originally proposed and later revised by Polson and colleagues [294, 376] as an attempt to introduce psychological theory into the informal and subjective walkthrough technique.

The origin of the cognitive walkthrough approach to evaluation is the code walkthrough familiar in software engineering. Walkthroughs require a detailed review of a sequence of actions. In the code walkthrough, the sequence represents a segment of the program code that is stepped through by the reviewers to check certain characteristics (for example, that coding style is adhered to, conventions for spelling variables versus procedure calls, and to check that system-wide invariants are not violated). In the cognitive walkthrough, the sequence of actions refers to the steps that an interface will require a user to perform in order to accomplish some known task. The evaluators then 'step through' that action sequence to check it for potential usability problems. Usually, the main focus of the cognitive walkthrough is to establish how easy a system is to learn. More specifically, the focus is on learning through exploration. Experience shows that many users prefer to learn how to use a system by exploring its functionality hands on, and not after sufficient training or examination of a user's manual. So the checks that are made during the walkthrough ask questions that address this exploratory learning. To do this, the evaluators go through each step in the task and provide a 'story' about why that step is or is not good for a new user. To do a walkthrough (the term walkthrough from now on refers to the cognitive walkthrough, and not to any other kind of walkthrough), you need four things:

1. A specification or prototype of the system. It doesn't have to be complete, but it should be fairly detailed. Details such as the location and wording for a menu can make a big difference.
2. A description of the task the user is to perform on the system. This should be a representative task that most users will want to do.
3. A complete, written list of the actions needed to complete the task with the proposed system.
4. An indication of who the users are and what kind of experience and knowledge the evaluators can assume about them.

Given this information, the evaluators step through the action sequence (identified in item 3 above) to critique the system and tell a believable story about its usability. To do this, for each action, the evaluators try to answer the following four questions for each step in the action sequence.

1. **Is the effect of the action the same as the user's goal at that point?** Each user action will have a specific effect within the system. Is this effect the same as what the user is trying to achieve at this point? For example, if the effect of the action is to save a document, is 'saving a document' what the user wants to do?
2. **Will users see that the action is available?** Will users see the button or menu item, for example, that is used to produce the action? This is *not* asking whether they will recognize that the button is the one they want. This is merely asking whether

it is visible to them at the time when they will need to use it. Instances where the answer to this question might be 'no' are, for example, where a VCR remote control has a covered panel of buttons or where a menu item is hidden away in a submenu.

3. **Once users have found the correct action, will they know it is the one they need?** This complements the previous question. It is one thing for a button or menu item to be visible, but will the user recognize that it is the one he is looking for to complete his task? Where the previous question was about the visibility of the action, this one is about whether its meaning and effect is clear.
4. **After the action is taken, will users understand the feedback they get?** If you now assume that the user did manage to achieve the correct action, will he know that he has done so? Will the feedback given be sufficient confirmation of what has actually happened? This is the completion of the execution–evaluation interaction cycle (see Chapter 3). In order to determine if they have accomplished their goal, users need appropriate feedback.

It is vital to document the cognitive walkthrough to keep a record of what is good and what needs improvement in the design. It is therefore a good idea to produce some standard evaluation forms for the walkthrough. The cover form would list the information in items 1–4 in our first list above, as well as identifying the date and time of the walkthrough and the names of the evaluators. Then for each action (from item 3 on the cover form), a separate standard form is filled out that answers each of the four questions in our second list above. Any negative answer for any of the questions for any particular action should be documented on a separate usability problem report sheet. This problem report sheet should indicate the system being built (the version, if necessary), the date, the evaluators and a detailed description of the usability problem. It is also useful to indicate the severity of the problem, that is whether the evaluators think this problem will occur often, and how serious it will be for the users. This information will help the designers to decide priorities for correcting the design, since it is not always possible to fix every problem.

Example: programming a video recorder by remote control

We can illustrate how the walkthrough method works using a simple example. Imagine we are designing a remote control for a video recorder (VCR) and are interested in the task of programming the VCR to do timed recordings. Our initial design is shown in Figure 9.1. The picture on the left illustrates the handset in normal use, the picture on the right after the timed record button has been pressed. The VCR allows the user to program up to three timed recordings in different 'streams'. The next available stream number is automatically assigned. We want to know whether our design supports the user's task. We begin by identifying a representative task.

Program the video to time-record a program starting at 18.00 and finishing at 19.15 on channel 4 on 24 February 2005.

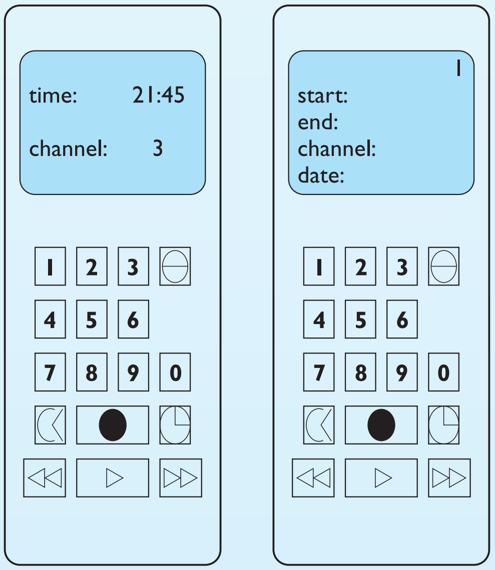


Figure 9.1 An initial remote control design

We will assume that the user is familiar with VCRs but not with this particular design.

The next step in the walkthrough is to identify the action sequence for this task. We specify this in terms of the user's action (UA) and the system's display or response (SD). The initial display is as the left-hand picture in Figure 9.1.

UA 1: Press the 'timed record' button
SD 1: Display moves to timer mode. Flashing cursor appears after 'start:'
UA 2: Press digits 1 8 0 0
SD 2: Each digit is displayed as typed and flashing cursor moves to next position
UA 3: Press the 'timed record' button
SD 3: Flashing cursor moves to 'end:'
UA 4: Press digits 1 9 1 5
SD 4: Each digit is displayed as typed and flashing cursor moves to next position
UA 5: Press the 'timed record' button
SD 5: Flashing cursor moves to 'channel:'
UA 6: Press digit 4
SD 6: Digit is displayed as typed and flashing cursor moves to next position
UA 7: Press the 'timed record' button
SD 7: Flashing cursor moves to 'date:'
UA 8: Press digits 2 4 0 2 0 5
SD 8: Each digit is displayed as typed and flashing cursor moves to next position
UA 9: Press the 'timed record' button
SD 9: Stream number in top right-hand corner of display flashes
UA 10: Press the 'transmit' button
SD 10: Details are transmitted to video player and display returns to normal mode

Having determined our action list we are in a position to proceed with the walkthrough. For each action (1–10) we must answer the four questions and tell a story about the usability of the system. Beginning with UA 1:

UA 1: Press the 'timed record' button
Question 1: Is the effect of the action the same as the user's goal at that point?
The timed record button initiates timer programming. It is reasonable to assume that a user familiar with VCRs would be trying to do this as his first goal.
Question 2: Will users see that the action is available?
The 'timed record' button is visible on the remote control.
Question 3: Once users have found the correct action, will they know it is the one they need?
It is not clear which button is the 'timed record' button. The icon of a clock (fourth button down on the right) is a possible candidate but this could be interpreted as a button to change the time. Other possible candidates might be the fourth button down on the left or the filled circle (associated with record). In fact, the icon of the clock is the correct choice but it is quite possible that the user would fail at this point. This identifies a potential usability problem.
Question 4: After the action is taken, will users understand the feedback they get?
Once the action is taken the display changes to the timed record mode and shows familiar headings (start, end, channel, date). It is reasonable to assume that the user would recognize these as indicating successful completion of the first action.

So we find we have a potential usability problem relating to the icon used on the 'timed record' button. We would now have to establish whether our target user group could correctly distinguish this icon from others on the remote.

The analysis proceeds in this fashion, with a walkthrough form completed for each action. We will leave the rest of the walkthrough for you to complete as an exercise. What other usability problems can you identify with this design?

9.3.2 Heuristic evaluation

A heuristic is a guideline or general principle or rule of thumb that can guide a design decision or be used to critique a decision that has already been made. *Heuristic evaluation*, developed by Jakob Nielsen and Rolf Molich, is a method for structuring the critique of a system using a set of relatively simple and general heuristics. Heuristic evaluation can be performed on a design specification so it is useful for evaluating early design. But it can also be used on prototypes, storyboards and fully functioning systems. It is therefore a flexible, relatively cheap approach. Hence it is often considered a *discount usability* technique.

The general idea behind heuristic evaluation is that several evaluators independently critique a system to come up with potential usability problems. It is important that there be several of these evaluators and that the evaluations be done independently. Nielsen's experience indicates that between three and five evaluators is sufficient, with five usually resulting in about 75% of the overall usability problems being discovered.

To aid the evaluators in discovering usability problems, a set of 10 heuristics are provided. The heuristics are related to *principles* and *guidelines* (see Chapter 7). These can be supplemented where required by heuristics that are specific to the particular domain. So, for example, if the system is for synchronous group communication, one might add 'awareness of other users' as a heuristic. Although Nielsen recommends the use of these 10 as providing the most effective coverage of the most common usability problems, other rules, such as those discussed in Chapter 7, could also be used.

Each evaluator assesses the system and notes violations of any of these heuristics that would indicate a potential usability problem. The evaluator also assesses the severity of each usability problem, based on four factors: how common is the problem, how easy is it for the user to overcome, will it be a one-off problem or a persistent one, and how seriously will the problem be perceived? These can be combined into an overall severity rating on a scale of 0–4:

0 = I don't agree that this is a usability problem at all
1 = Cosmetic problem only: need not be fixed unless extra time is available on project
2 = Minor usability problem: fixing this should be given low priority
3 = Major usability problem: important to fix, so should be given high priority
4 = Usability catastrophe: imperative to fix this before product can be released (Nielsen)

Nielsen's ten heuristics are:

1. **Visibility of system status** Always keep users informed about what is going on, through appropriate feedback within reasonable time. For example, if a system operation will take some time, give an indication of how long and how much is complete.
2. **Match between system and the real world** The system should speak the user's language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in natural and logical order.
3. **User control and freedom** Users often choose system functions by mistake and need a clearly marked 'emergency exit' to leave the unwanted state without having to go through an extended dialog. Support undo and redo.
4. **Consistency and standards** Users should not have to wonder whether words, situations or actions mean the same thing in different contexts. Follow platform conventions and accepted standards.
5. **Error prevention** Make it difficult to make errors. Even better than good error messages is a careful design that prevents a problem from occurring in the first place.
6. **Recognition rather than recall** Make objects, actions and options visible. The user should not have to remember information from one part of the dialog to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.
7. **Flexibility and efficiency of use** Allow users to tailor frequent actions. Accelerators – unseen by the novice user – may often speed up the interaction for the expert user to such an extent that the system can cater to both inexperienced and experienced users.

8. **Aesthetic and minimalist design** Dialogs should not contain information that is irrelevant or rarely needed. Every extra unit of information in a dialog competes with the relevant units of information and diminishes their relative visibility.
9. **Help users recognize, diagnose and recover from errors** Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.
10. **Help and documentation** Few systems can be used with no instructions so it may be necessary to provide help and documentation. Any such information should be easy to search, focussed on the user's task, list concrete steps to be carried out, and not be too large.

Once each evaluator has completed their separate assessment, all of the problems are collected and the mean severity ratings calculated. The design team will then determine the ones that are the most important and will receive attention first.

9.3.3 Model-based evaluation

A third expert-based approach is the use of models. Certain cognitive and design models provide a means of combining design specification and evaluation into the same framework. These are discussed in detail in Chapter 12. For example, the GOMS (goals, operators, methods and selection) model predicts user performance with a particular interface and can be used to filter particular design options. Similarly, lower-level modeling techniques such as the keystroke-level model provide predictions of the time users will take to perform low-level physical tasks.

Design methodologies, such as design rationale (see Chapter 6), also have a role to play in evaluation at the design stage. Design rationale provides a framework in which design options can be evaluated. By examining the criteria that are associated with each option in the design, and the evidence that is provided to support these criteria, informed judgments can be made in the design.

Dialog models can also be used to evaluate dialog sequences for problems, such as unreachable states, circular dialogs and complexity. Models such as state transition networks are useful for evaluating dialog designs prior to implementation. These are discussed in detail in Chapter 16.

9.3.4 Using previous studies in evaluation

Experimental psychology and human–computer interaction between them possess a wealth of experimental results and empirical evidence. Some of this is specific to a particular domain, but much deals with more generic issues and applies in a variety of situations. Examples of such issues are the usability of different menu types, the recall of command names, and the choice of icons.

A final approach to expert evaluation exploits this inheritance, using previous results as evidence to support (or refute) aspects of the design. It is expensive to repeat experiments continually and an expert review of relevant literature can avoid the need to do so. It should be noted that experimental results cannot be expected to hold arbitrarily across contexts. The reviewer must therefore select evidence carefully, noting the experimental design chosen, the population of participants used, the analyses performed and the assumptions made. For example, an experiment testing the usability of a particular style of help system using novice participants may not provide accurate evaluation of a help system designed for expert users. The review should therefore take account of both the similarities and the differences between the experimental context and the design under consideration. This is why this is an *expert* review: expertise in the area is required to ensure that correct assumptions are made.

9.4 EVALUATION THROUGH USER PARTICIPATION

The techniques we have considered so far concentrate on evaluating a design or system through analysis by the designer, or an expert evaluator, rather than testing with actual users. However, useful as these techniques are for filtering and refining the design, they are not a replacement for actual usability testing with the people for whom the system is intended: the users. In this section we will look at a number of different approaches to evaluation through user participation. These include empirical or experimental methods, observational methods, query techniques, and methods that use physiological monitoring, such as eye tracking and measures of heart rate and skin conductance.

User participation in evaluation tends to occur in the later stages of development when there is at least a working prototype of the system in place. This may range from a simulation of the system's interactive capabilities, without its underlying functionality (for example, the *Wizard of Oz* technique, which is discussed in Chapter 6, through a basic functional prototype to a fully implemented system. However, some of the methods discussed can also contribute to the earlier design stages, such as requirements capture, where observation and surveying users are important (see Chapter 13).

9.4.1 Styles of evaluation

Before we consider some of the techniques that are available for evaluation with users, we will distinguish between two distinct evaluation styles: those performed under laboratory conditions and those conducted in the work environment or 'in the field'.

Laboratory studies

In the first type of evaluation studies, users are taken out of their normal work environment to take part in controlled tests, often in a specialist usability laboratory (although the 'lab' may simply be a quiet room). This approach has a number of benefits and disadvantages.

A well-equipped usability laboratory may contain sophisticated audio/visual recording and analysis facilities, two-way mirrors, instrumented computers and the like, which cannot be replicated in the work environment. In addition, the participant operates in an interruption-free environment. However, the lack of context – for example, filing cabinets, wall calendars, books or interruptions – and the unnatural situation may mean that one accurately records a situation that never arises in the real world. It is especially difficult to observe several people cooperating on a task in a laboratory situation, as interpersonal communication is so heavily dependent on context (see Section 9.4.2).

There are, however, some situations where laboratory observation is the only option, for example, if the system is to be located in a dangerous or remote location, such as a space station. Also some very constrained single-user tasks may be adequately performed in a laboratory. Finally, and perhaps most commonly, we may deliberately want to manipulate the context in order to uncover problems or observe less used procedures, or we may want to compare alternative designs within a controlled context. For these types of evaluation, laboratory studies are appropriate.

Field studies

The second type of evaluation takes the designer or evaluator out into the user's work environment in order to observe the system in action. Again this approach has its pros and cons.

High levels of ambient noise, greater levels of movement and constant interruptions, such as phone calls, all make field observation difficult. However, the very 'open' nature of the situation means that you will observe interactions between systems and between individuals that would have been missed in a laboratory study. The context is retained and you are seeing the user in his 'natural environment'. In addition, some activities, such as those taking days or months, are impossible to study in the laboratory (though difficult even in the field).

On balance, field observation is to be preferred to laboratory studies as it allows us to study the interaction as it occurs in actual use. Even interruptions are important as these will expose behaviors such as saving and restoring state during a task. However, we should remember that even in field observations the participants are likely to be influenced by the presence of the analyst and/or recording equipment, so we always operate at a slight remove from the natural situation, a sort of Heisenberg uncertainty principle.

This is, of course, a generalization: there are circumstances, as we have noted, in which laboratory testing is necessary and desirable. In particular, controlled experiments can be useful for evaluation of specific interface features, and must normally be conducted under laboratory conditions. From an economic angle, we need to weigh the costs of establishing recording equipment in the field, and possibly disrupting the actual work situation, with the costs of taking one or more participants away from their jobs into the laboratory. This balance is not at all obvious and any study must weigh the loss of contextual information against the increased costs and difficulty of field studies.

9.4.2 Empirical methods: experimental evaluation

One of the most powerful methods of evaluating a design or an aspect of a design is to use a controlled experiment. This provides empirical evidence to support a particular claim or hypothesis. It can be used to study a wide range of different issues at different levels of detail.

Any experiment has the same basic form. The evaluator chooses a hypothesis to test, which can be determined by measuring some attribute of participant behavior. A number of experimental conditions are considered which differ only in the values of certain controlled variables. Any changes in the behavioral measures are attributed to the different conditions. Within this basic form there are a number of factors that are important to the overall reliability of the experiment, which must be considered carefully in experimental design. These include the participants chosen, the variables tested and manipulated, and the hypothesis tested.

Participants

The choice of participants is vital to the success of any experiment. In evaluation experiments, participants should be chosen to match the expected user population as closely as possible. Ideally, this will involve experimental testing with the actual users but this is not always possible. If participants are not actual users, they should be chosen to be of a similar age and level of education as the intended user group. Their experience with computers in general, and with systems related to that being tested, should be similar, as should their experience or knowledge of the task domain. It is no good testing an interface designed to be used by the general public on a participant set made up of computer science undergraduates: they are simply not representative of the intended user population.

A second issue relating to the participant set is the sample size chosen. Often this is something that is determined by pragmatic considerations: the availability of participants is limited or resources are scarce. However, the sample size must be large enough to be considered to be representative of the population, taking into account the design of the experiment and the statistical methods chosen.

Nielsen and Landauer [264] suggest that usability testing with a single participant will find about a third of the usability problems, and that there is little to be gained from testing with more than five. While this may be true of observational studies where the aim is simply to uncover usability issues, it is not possible to discover much about the extent of usability problems from such small numbers. Certainly, if the intention is to run a controlled experiment and perform statistical analysis on the results, at least twice this number is recommended.

Variables

Experiments manipulate and measure variables under controlled conditions, in order to test the hypothesis. There are two main types of variable: those that are ‘manipulated’ or changed (known as the independent variables) and those that are measured (the dependent variables).

Independent variables are those elements of the experiment that are manipulated to produce different conditions for comparison. Examples of independent variables in evaluation experiments are interface style, level of help, number of menu items and icon design. Each of these variables can be given a number of different values; each value that is used in an experiment is known as a *level* of the variable. So, for example, an experiment that wants to test whether search speed improves as the number of menu items decreases may consider menus with five, seven, and ten items. Here the independent variable, number of menu items, has three levels.

More complex experiments may have more than one independent variable. For example, in the above experiment, we may suspect that the speed of the user’s response depends not only on the number of menu items but also on the choice of commands used on the menu. In this case there are two independent variables. If there were two sets of command names (that is, two levels), we would require six experimental conditions to investigate all the possibilities (three levels of menu size × two levels of command names).

Dependent variables, on the other hand, are the variables that can be measured in the experiment, their value is ‘dependent’ on the changes made to the independent variable. In the example given above, this would be the speed of menu selection. The dependent variable must be measurable in some way, it must be affected by the independent variable, and, as far as possible, unaffected by other factors. Common choices of dependent variable in evaluation experiments are the time taken to complete a task, the number of errors made, user preference and the quality of the user’s performance. Obviously, some of these are easier to measure objectively than others. However, the more subjective measures can be applied against predetermined scales, and can be very important factors to consider.

Hypotheses

A hypothesis is a prediction of the outcome of an experiment. It is framed in terms of the independent and dependent variables, stating that a variation in the independent variable will cause a difference in the dependent variable. The aim of the experiment is to show that this prediction is correct. This is done by disproving the null hypothesis, which states that there is no difference in the dependent variable between the levels of the independent variable. The statistical measures described below produce values that can be compared with various levels of significance. If a result is significant it shows, at the given level of certainty, that the differences measured would not have occurred by chance (that is, that the null hypothesis is incorrect).

Experimental design

In order to produce reliable and generalizable results, an experiment must be carefully designed. We have already looked at a number of the factors that the experimenter must consider in the design, namely the participants, the independent and dependent variables, and the hypothesis. The first phase in experimental design then is to choose the hypothesis: to decide exactly what it is you are trying to demonstrate. In doing this you are likely to clarify the independent and dependent variables, in that you will have identified what you are going to manipulate and what change you expect. If your hypothesis does not clearly identify these variables then you need to rethink it. At this stage you should also consider your participants: how many are available and are they representative of the user group?

The next step is to decide on the *experimental method* that you will use. There are two main methods: *between-subjects* and *within-subjects*. In a between-subjects (or *randomized*) design, each participant is assigned to a different condition. There are at least two conditions: the experimental condition (in which the variable has been manipulated) and the control, which is identical to the experimental condition except for this manipulation. This control serves to ensure that it is the manipulation that is responsible for any differences that are measured. There may, of course, be more than two groups, depending on the number of independent variables and the number of levels that each variable can take.

The advantage of a between-subjects design is that any learning effect resulting from the user performing in one condition and then the other is controlled: each user performs under only one condition. The disadvantages are that a greater number of participants are required, and that significant variation between the groups can negate any results. Also, individual differences between users can bias the results. These problems can be handled by a careful selection of participants, ensuring that all are representative of the population and by matching participants between groups.

The second experimental design is within-subjects (or *repeated measures*). Here each user performs under each different condition. This design can suffer from transfer of learning effects, but this can be lessened if the order in which the conditions are tackled is varied between users, for example, group A do first condition followed by second and group B do second condition followed by first. Within-subjects is less costly than between-subjects, since fewer users are required, and it can be particularly effective where learning is involved. There is also less chance of effects from variation between participants.

The choice of experimental method will depend on the resources available, how far learning transfer is likely or can be controlled, and how representative the participant group is considered to be. A popular compromise, in cases where there is more than one independent variable, is to devise a mixed design where one variable is placed between-groups and one within-groups. So, returning to our example of the menu design, the participants would be split into two groups, one for each command set, but each group would perform in three conditions, corresponding to the three possible levels of the number of menu items.

Once we have determined the hypothesis we are trying to test, the variables we are studying, the participants at our disposal, and the design that is most appropriate, we have to decide how we are going to analyze the results we record. There are a number of statistical tests available, and the choice of test is vital to the success of the experiment. Different tests make different assumptions about the data and if an inappropriate test is chosen, the results can be invalid. The next subsection discusses the factors to consider in choosing a statistical test and surveys the most common statistical measures available.

Statistical measures

The first two rules of statistical analysis are to *look* at the data and to *save* the data. It is easy to carry out statistical tests blindly when a glance at a graph, histogram or table of results would be more instructive. In particular, looking at the data can expose *outliers*, single data items that are very different from the rest. Outliers are often the result of a transcription error or a freak event not connected to the experiment. For example, we notice that one participant took three times as long as everyone else to do a task. We investigate and discover that the participant had been suffering from flu on the day of the experiment. Clearly, if the participant's data were included it would bias the results.

Saving the data is important, as we may later want to try a different analysis method. It is all too common for an experimenter to take some averages or otherwise tabulate results, and then throw away the original data. At worst, the remaining statistics can be useless for statistical purposes, and, at best, we have lost the ability to trace back odd results to the original data, as, for example, we want to do for outliers.

Our choice of statistical analysis depends on the type of data and the questions we want to answer. It is worth having important results checked by an experienced statistician, but in many situations standard tests can be used.

Variables can be classified as either *discrete variables* or *continuous variables*. A discrete variable can only take a finite number of values or *levels*, for example, a screen color that can be red, green or blue. A continuous variable can take any value (although it may have an upper or lower limit), for example a person's height or the time taken to complete a task. A special case of continuous data is when they are *positive*, for example a response time cannot be negative. A continuous variable can be rendered discrete by clumping it into classes, for example we could divide heights into short (<5 ft (1.5 m)), medium (5–6 ft (1.5–1.8 m)) and tall (>6 ft (1.8 m)). In many interface experiments we will be testing one design against another. In these cases the independent variable is usually discrete.

The dependent variable is the measured one and subject to random experimental variation. In the case when this variable is continuous, the random variation may take a special form. If the form of the data follows a known *distribution* then special and more powerful statistical tests can be used. Such tests are called *parametric tests* and the most common of these are used when the variation follows the *normal distribution*. This means that if we plot a histogram of the random errors, they will

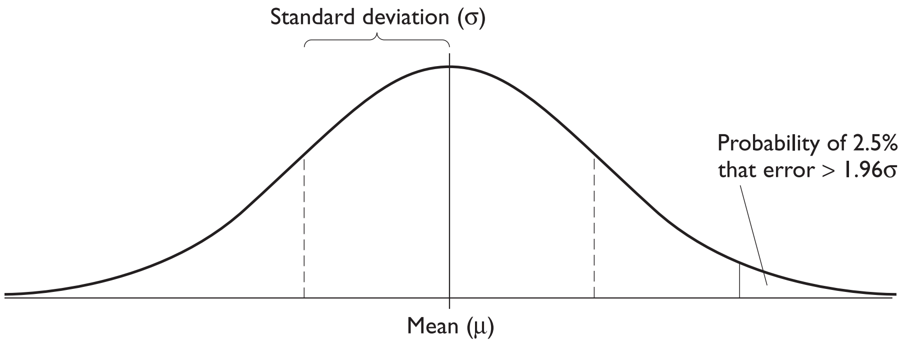


Figure 9.2 Histogram of normally distributed errors

form the well-known bell-shaped graph (Figure 9.2). Happily, many of these tests are fairly *robust*, that is they give reasonable results even when the data are not precisely normal. This means that you need not worry too much about checking normality during early analysis.

There are ways of checking whether data are really normal, but for these the reader should consult a statistics book, or a professional statistician. However, as a general rule, if data can be seen as the sum or average of many small *independent* effects they are likely to be normal. For example, the time taken to complete a *complex* task is the sum of the times of all the minor tasks of which it is composed. On the other hand, a subjective rating of the usability of an interface will not be normal. Occasionally data can be *transformed* to become approximately normal. The most common is the log-transformation, which is used for positive data with near-zero values. As a log-transformation has little effect when the data are clustered well away from zero, many experimenters habitually log-transform. However, this practice makes the results more difficult to interpret and is not recommended.

When we cannot assume that data are normally distributed, we must often resort to *non-parametric* tests. These are statistical tests that make no assumptions about the particular distribution and are usually based purely on the ranking of the data. That is, each item of a data set (for example, 57, 32, 61, 49) is reduced to its rank (3, 1, 4, 2), before analysis begins. Because non-parametric tests make fewer assumptions about the data than parametric tests, and are more resistant to outliers, there is less danger of getting spurious results. However, they are less *powerful* than the corresponding parametric tests. This means that, given the same set of data, a parametric test might detect a difference that the non-parametric test would miss.

A third sort of test is the contingency table, where we classify data by several discrete attributes and then count the number of data items with each attribute combination.

Table 9.1 lists some of the standard tests categorized by the form of independent and dependent variables (discrete/continuous/normal). Normality is not an issue

Table 9.1 Choosing a statistical technique

Independent variable	Dependent variable	
Parametric		
Two valued	Normal	Student's t test on difference of means
Discrete	Normal	ANOVA (ANalysis Of VAriance)
Continuous	Normal	Linear (or non-linear) regression factor analysis
Non-parametric		
Two valued	Continuous	Wilcoxon (or Mann–Whitney) rank-sum test
Discrete	Continuous	Rank-sum versions of ANOVA
Continuous	Continuous	Spearman's rank correlation
Contingency tests		
Two valued	Discrete	No special test, see next entry
Discrete	Discrete	Contingency table and chi-squared test
Continuous	Discrete	(Rare) Group independent variable and then as above

for the independent variable, but a special case is when it is discrete with only two values, for example comparing two systems. We cannot describe all the techniques here; for this you should use a standard statistics text, such as one of those recommended in the reading list. The table is only intended to guide you in your choice of test.

An extensive and accurate analysis is no use if it answers the wrong question. Examples of questions one might ask about the data are as follows:

Is there a difference? For example, is one system better than another? Techniques that address this are called *hypothesis testing*. The answers to this question are not simply yes/no, but of the form: 'we are 99% certain that selection from menus of five items is faster than that from menus of seven items'.

How big is the difference? For example, 'selection from five items is 260 ms faster than from seven items'. This is called *point estimation*, often obtained by averages.

How accurate is the estimate? For example, 'selection is faster by 260 ± 30 ms'. Statistical answers to this are in the form of either measures of variation such as the *standard deviation* of the estimate, or *confidence intervals*. Again, the answers one obtains are probabilistic: 'we are 95% certain that the difference in response time is between 230 and 290 ms'.

The experimental design issues we have discussed have been principally addressed at the first question. However, most of the statistical techniques listed above, both parametric and non-parametric, give some answer to one or both of the other questions.

Example of non-parametric statistics

We will not see an example of the use of non-parametric statistics later, so we will go through a small example here. Imagine we had the following data for response times under two conditions:

condition A:	33, 42, 25, 79, 52
condition B:	87, 65, 92, 93, 91, 55

We gather the data together and sort them into order: 25, 33, 42, . . . , 92, 93. We then substitute for each value its rank in the list: 25 becomes 1, 33 becomes 2, etc. The transformed data are then

condition A:	2, 3, 1, 7, 4
condition B:	8, 6, 10, 11, 9, 5

Tests are then carried out on the data. For example, to test whether there is any difference between the two conditions we can use the *Wilcoxon test*. To do this, we take each condition and calculate the sum of ranks, and subtract the least value it could have (that is, $1 + 2 + 3 + 4 + 5 = 15$ for condition A, $1 + 2 + 3 + 4 + 5 + 6 = 21$ for condition B), giving the statistic U:

	rank sum		least		U
condition A:	$(2 + 3 + 1 + 7 + 4)$	$-$	15	$=$	2
condition B:	$(8 + 6 + 10 + 11 + 9 + 5)$	$-$	21	$=$	28

In fact, the sum of these two U statistics, $2 + 28 = 30$, is the product of the number of data values in each condition 5×6. This will always happen and so one can always get away with calculating only one of the U. Finally, we then take the smaller of two U values and compare it with a set of *critical values* in a book of statistical tables, to see if it is unusually small. The table is laid out dependent on the number of data values in each condition (five and six). The critical value at the 5% level turns out to be 3. As the smallest statistic is smaller than this, we can *reject the null hypothesis* and conclude that there is likely to be a difference between the conditions. To be precise, it says that there is only a 1 in 20 (5%) chance that the data happened by chance. In fact the test is right – the authors constructed random data in the range 1–100 and then subtracted 10 from each of the values in condition A.

An example: evaluating icon designs

Imagine you are designing a new interface to a document-processing package, which is to use icons for presentation. You are considering two styles of icon design and you wish to know which design will be easier for users to remember. One set of icons uses naturalistic images (based on a paper document metaphor), the other uses abstract images (see Figure 9.3). How might you design an experiment to help you decide which style to use?

The first thing you need to do is form a hypothesis: what do you consider to be the likely outcome? In this case, you might expect the natural icons to be easier to recall since they are more familiar to users. We can therefore form the following hypothesis:

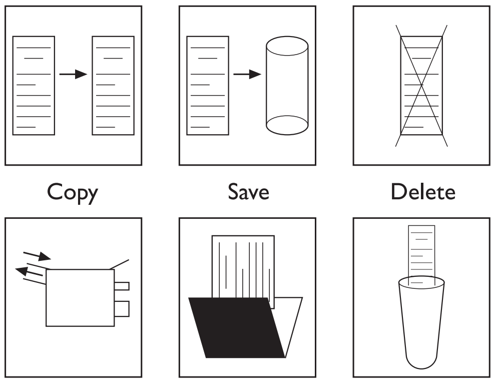


Figure 9.3 Abstract and concrete icons for file operations

Users will remember the natural icons more easily than the abstract ones.

The null hypothesis in this case is that there will be no difference between recall of the icon types.

This hypothesis clearly identifies the independent variable for our experiment: we are varying the style of icon. The independent variable has two levels: natural and abstract. However, when we come to consider the dependent variable, things are not so obvious. We have expressed our hypothesis in terms of users being able to remember *more easily*. How can we measure this? First we need to clarify exactly what we mean by the phrase *more easily*: are we concerned with the user's performance in terms of accurate recall or in terms of speed, for example, or are we looking at more subjective measures like user preference? In this example, we will assume that the speed at which a user can accurately select an icon is an indication of how easily it is remembered. Our dependent variables are therefore the number of mistakes in selection and the time taken to select an icon.

Of course, we need to control the experiment so that any differences we observe are clearly attributable to the independent variable, and so that our measurements of the dependent variables are comparable. To do this, we provide an interface that is identical in every way except for the icon design, and a selection task that can be repeated for each condition. The latter could be either a naturalistic task (such as producing a document) or a more artificial task in which the user has to select the appropriate icon to a given prompt. The second task has the advantage that it is more controlled (there is little variation between users as to how they will perform the task) and it can be varied to avoid transfer of learning. Before performing the selection task, the users will be allowed to learn the icons in controlled conditions: for example, they may be given a fixed amount of time to learn the icon meanings.

The next stage is to decide upon an experimental method. This may depend on the participants that are available, but in this case we will assume that we have sufficient participants from the intended user group. A between-subjects experiment would remove any learning effect for individual participants, but it would be more difficult

Table 9.2 Example experimental results – completion times

Participant number	Presentation order	(1) Natural (s)	(2) Abstract (s)	(3) Participant mean	(4) Natural (1)–(3)	(5) Abstract (2)–(3)
1	AN	656	702	679	−23	23
2	AN	259	339	299	−40	40
3	AN	612	658	635	−23	23
4	AN	609	645	627	−18	18
5	AN	1049	1129	1089	−40	40
6	NA	1135	1179	1157	−22	22
7	NA	542	604	573	−31	31
8	NA	495	551	523	−28	28
9	NA	905	893	899	6	−6
10	NA	715	803	759	−44	44
mean (μ)		698	750	724	−26	26
s.d. (σ)		265	259	262	14	14
			s.e.d. 117		s.e. 4.55	
Student's *t*			0.32 (n.s.)		5.78 (p<1%, two tailed)	

to control for variation in learning style between participants. On balance, therefore, a within-subjects design is preferred, with order of presentation controlled.

So all that remains is to finalize the details of our experiment, given the constraints imposed by these choices. We devise two interfaces composed of blocks of icons, one for each condition. The user is presented with a task (say 'delete a document') and is required to select the appropriate icon. The selection task comprises a set of such presentations. In order to avoid learning effects from icon position, the placing of icons in the block can be randomly varied on each presentation. Each user performs the selection task under each condition. In order to avoid transfer of learning, the users are divided into two groups with each group taking a different starting condition. For each user, we measure the time taken to complete the task and the number of errors made.

Finally, we must analyze our results. Table 9.2 shows a possible set of results for ten participants.[1] The first five had the abstract icons presented first (order AN), and the last five had the natural icons presented first (order NA). Columns (1) and (2) in the table show the completion times for the task using natural and abstract icons respectively. As the times are the result of lots of presentations, we will assume that they are normally distributed. The main independent variable, the icon type, is two valued, suggesting we can use a simple difference of means with Student's *t* test (Table 9.1). In fact, because we have used a *within-subjects* design, there is another independent variable we have to take into account – the participant. This means we

1 Note that these are fabricated results for the purposes of exposition and this is a rather small sample set for real purposes.

have more than one discrete independent variable, and referring again to Table 9.1, we see that this implies we should use *analysis of variance* (*ANOVA*). A full analysis of variance is quite complex, and is ideally done with the aid of a statistics package. However, this experiment is particularly simple, so we can use a simplified analysis.

Look at columns (2) and (3) of Table 9.2. The completion times range from less than 5 minutes (participant 2) to nearly 20 minutes (participant 6), showing a wide variation between individuals. This wide variation emphasizes the importance of the *within*-subjects design. To see how this affects the results, we will first try to analyze them ignoring the fact that each participant performed under each condition. At the end of the table, the mean and standard deviation have been calculated for each condition. These means can then be compared using Student's *t* test. The difference between the means is 52 seconds, but the *standard error of the difference* (s.e.d.) is 117. This is calculated as follows:

$$\text{s.e.d.} = \sqrt{\frac{\sigma_N^2}{n_N} + \frac{\sigma_A^2}{n_A}} = \sqrt{\frac{265^2}{10} + \frac{259^2}{10}} = 117.2$$

where σ_N and σ_A are the standard deviations (s.d.) of the two conditions, and n_N and n_A are the number of data items in each condition (10 in each). The s.e.d. is a measure of the expected variability of the difference between the means, and as we see the actual difference is well within this random variation. Testing the ratio 52/117 against tables of Student's *t* distribution indeed shows that this is not significant.

However, if we glance down the table, we see that in almost every case the time taken with the abstract icons is greater than the time taken for the natural icons. That is, the data seem to support our claim that natural icons are better than abstract ones, but the wide variation between individuals has hidden the effect.

A more sophisticated analysis, a special case of ANOVA, can expose the difference. Looking back at the table, column (3) shows, for each participant, the average of the time they took under the two conditions. This participant mean is then subtracted from the data for each condition, yielding columns (4) and (5). These columns show the effect of the icon design *once the differences between participants have been removed*. The two columns are redundant as they always add up to zero. They show that in all but one case (participant 9) the natural icons are faster than the abstract ones.

Even a non-parametric test would show this as a significant difference at the 5% level, but the use of a *t* test is more precise. We can take either column and see that the column average 26 is much greater than the standard error ($14.4/\sqrt{10}$). The ratio (mean/s.e.) is compared with the Student's *t* table (in statistical tables) using nine degrees of freedom (10 values minus 1 for the mean), and is indeed far greater than the 1% level (3.250); that is, the likelihood of getting our results by chance is less than 1 in 100. So, we reject the null hypothesis that there is no difference and conclude that natural icons are more easily remembered than abstract ones.

In fact, the last statement is not quite correct. What we have shown is that in this experiment natural icons are more *rapidly* remembered. Possibly, if we go on to analyze the errors, these may present a different story. If these error figures were quite large (say 15 errors or more per condition), then we may be able to assume these are normal and use ANOVA. If not, we can either use non-parametric tests, or make use of special tests based on the *binomial distribution*. We will not perform these analyses here. Possibly, looking at the errors we may find that the natural icons have *more* errors – it could well be that they are more rapidly, but less accurately, remembered. It is always worth keeping in mind the difference between the intended purpose of the experiment (to see which is better remembered) and the actual measurements (speed and accuracy).

Finally, one ought to look carefully at the experimental results to see whether there is any other effect that might confuse the results. The graphical presentation of results will help with this, possibly highlighting odd clumps in the data or other irregularities. In this experiment we may want to check to see if there has been any significant *transfer effect* between the first and second condition for each participant. The second set may be faster as the participants are more practiced, or possibly the second set may be slower as learning a second set of icons may be confusing. This will not matter if the effect is uniform – say they always are 15 seconds slower on the second test. But there may be systematic effects. For example, seeing the natural icons first might make it more difficult to learn the abstract ones, but not vice versa. If this were the case, our observed effect may be about the interference between the icon sets, rather than that one is better than the other.

Worked exercise *Design an experiment to test whether adding color coding to an interface will improve accuracy. Identify your hypothesis, participant group, dependent and independent variables, experimental design, task and analysis approach.*

Answer The following is only an example of the type of experiment that might be devised.

Participants Taken from user population.

Hypothesis Color coding will make selection more accurate.

IV (Independent Variable) Color coding.

DV (Dependent Variable) Accuracy measured as number of errors.

Design Between-groups to ensure no transfer of learning (or within-groups with appropriate safeguards if participants are scarce).

Task The interfaces are identical in each of the conditions, except that, in the second, color is added to indicate related menu items. Participants are presented with a screen of menu choices (ordered randomly) and verbally told what they have to select. Selection must be done within a strict time limit when the screen clears. Failure to select the correct item is deemed an error. Each presentation places items in new positions. Participants perform in one of the two conditions.

Analysis *t* test.

Studies of groups of users

So far we have considered the experimental evaluation of single-user systems. Experiments to evaluate elements of group systems bring additional problems. Given the complexities of human–human communication and group working, it is hardly surprising that experimental studies of groups and of groupware are more difficult than the corresponding single-user experiments already considered. For the purpose of discussion, let us assume that we are evaluating a shared application with video connections between the participants and consider some of the problems we will encounter.

The participant groups To organize, say, 10 experiments of a single-user system requires 10 participants. For an experiment involving groups of three, we will, of course, need 30 participants for the same number of experiments. In addition, experiments in group working are often longer than the single-user equivalents as we must allow time for the group to 'settle down' and some rapport to develop. This all means more disruption for participants and possibly more expense payments.

Arranging a mutually convenient slot when both participants and the equipment are available is no mean feat. Often the workstations being used in the experiment will be colleagues' personal systems, so we are trying to accommodate at least six people, not to mention the experimenters themselves.

Not surprisingly, many reports of group working involve only three or four groups. This is obviously a problem for statistical purposes, but not the primary obstacle.

The experimental task Choosing a suitable task is also difficult. We may want to test a variety of different task types: creative, structured, information passing, and so on. Also, the tasks must encourage active cooperation, either because the task requires consensus, or because information and control is distributed among the participants. Obviously, the task also depends on the nature of the groupware system: if it has several available channels, we want to encourage broad use. For example, in the case of shared application with video, it should not be possible (or at least not easy) to perform the task without using the application, otherwise we are simply investigating video conferencing.

Creative tasks such as 'write a short report on . . .' or 'write a research proposal' are often effective, in that the participants must reach agreement, and can be asked to produce their final report using the shared application. Design tasks are also used. For instance, in one experiment, users of the York Conferencer system (see Figure 14.2 in Section 14.4) were asked to redesign a bank layout. A picture of the current layout was used as a background for the spatially arranged electronic pinboard, and the participants made use of this to arrange comments and suggestions close to the features they referred to.

Decision games, as used in management courses, are designed to test and train cooperative activity. They often rely for their success on group coordination, not individual ability. An example of this is the desert survival task, where the participants are told that they have crashed in the desert. They are given a list of items to rank in order of importance for their survival: knife, plastic sheet, etc. The participants must produce *one* list between them, a single knowledgeable participant cannot 'go it alone'. A computerized version of the game of Diplomacy has also been used (see Figure 14.5 in Section 14.4) as it includes aspects of conflict as well as cooperation.

Finally, time-critical simulated process control tasks force a higher pace of interaction as the participants control different parts of the model. An example of this is ARKola [147], a simulated bottling plant, which was used at Xerox PARC to investigate the importance of background noise in complex cooperative control tasks.

Often the chosen task will require extra implementation effort, and in the case of games this may be extensive. This is obviously a strong factor in the choice of a suitable task.

Data gathering Even in a single-user experiment we may well use several video cameras as well as direct logging of the application. In a group setting this is replicated for each participant. So for a three-person group, we are trying to synchronize the recording of six or more video sources and three keystroke logs. To compound matters, these may be spread over different offices, or even different sites. The technical problems are clearly enormous. Four-into-one video recording is possible, storing a different image in each quadrant of the screen, but even this is insufficient for the number of channels we would like.

One way round this is to focus on the participants individually, recording, for each one, the video images that are being relayed as part of the system (assuming there is a video connection) and the sounds that the participant hears. These can then be synchronized with the particular participant's keystrokes and additional video observations. Thus, we can recreate the situation as it appeared *to the participant*. From this recording, we may not be able to interpret the other participants' actions, but at least we have a complete record for one.

Given sufficient recording equipment, this can be repeated for each participant. Happily, the level of synchronization required between participants is not as great as that required for each one individually. One can simply start the recorders' clocks at the same time, but not worry about sub-second accuracy between participants. The important thing is that we can, as it were, relive the experience for each individual.

Analysis In true experimental tradition, we would like to see statistical differences between experimental conditions. We saw earlier that individual differences made this difficult in single-user experiments. If anything, group variation is more extreme. Given randomly mixed groups, one group will act in a democratic fashion; in another, a particular pair will dominate discussion; in a third, one of the participants will act as coordinator, filtering the others' contributions. The level of variation is such that even catastrophic failures under one condition and fabulous successes in another may not always lead to statistically significant results.

As an example of this, imagine we have some quantitative measure of quality of output. We will almost certainly have to use non-parametric tests, so imagine we have found that all the groups under one condition obtained higher scores than any group under the other condition. We would need at least four in each condition to obtain even 5% significance (one tailed). If our results were only slightly less good, say one of the generally better groups performed poorly, we would then require at least five in each condition.

Now this example only considered one condition, and assumed the best possible results. In general, we would expect that the spread between groups within conditions would be greater, and we may want to test more conditions at once. Our 10 groups will have to increase rapidly to stand any chance of statistically significant results. However, we saw above that even gathering 10 experimental groups is a significant problem.

There are three possible solutions to this problem. First, one can use within-group experiments, having each group work under several conditions. We have, of course, the normal problems of such analysis, transfer effects and the like, but we also have more chance of cancelling out the group effect. Secondly, we can look to a micro-analysis of features like gaps between utterances. Such measures are more likely to fit a standard distribution, and thus one can use more powerful parametric tests. In addition, they may be more robust to the large-scale social differences between groups.

The third solution is to opt for a more anecdotal analysis, looking for critical incidents – for example, interesting events or breakdowns – in the data. The concepts and methods for analyzing conversation in Chapter 14 can be used to drive such an analysis. The advantage of this approach is that instead of regarding group differences as a 'problem', they can be included in the analysis. That is, we can begin to look for the systematic ways in which different group structures interact with the communications media and applications they use.

Of course, experiments can be analyzed using both quantitative and qualitative methods. Indeed, any detailed anecdotal analysis of the logs will indicate fruitful measures for statistical analysis. However, if the number of experimental groups is limited, attempts at controlled experiments may not be productive, and may effectively 'waste' the groups used in the control. Given the high costs of group-working experiments, one must choose conditions that are likely to give interesting results, even if statistical analysis proves impossible.

Field studies with groups There are, of course, problems with taking groups of users and putting them in an experimental situation. If the groups are randomly mixed, then we are effectively examining the process of group formation, rather than that of a normal working group. Even where a pre-existent group is used, excluding people from their normal working environment can completely alter their working patterns. For a new system, there may be no 'normal' workplace and all we can do is produce an artificial environment. However, even with a new system we have the choice of producing a 'good' experiment or a naturalistic setting. The traditions of experimental psychology are at odds with those of more qualitative sociological analysis.

It can be argued that group work can only be studied in context. Moving out of the real situation will alter the very nature of the work that is studied. Alternative approaches from the social sciences, such as ethnography, have therefore become popular, particularly in relation to studying group interaction. Ethnography involves very detailed recording of the interactions between people, their environment and each other. The ethnographer attempts to remain outside the situation being studied and does not impose a particular viewpoint on what is observed. This is very different from the experimental perspective with its hypothesis testing. Ethnography is discussed in more detail in Chapter 13.

9.4.3 Observational techniques

A popular way to gather information about actual use of a system is to observe users interacting with it. Usually they are asked to complete a set of predetermined tasks, although, if observation is being carried out in their place of work, they may be observed going about their normal duties. The evaluator watches and records the users' actions (using a variety of techniques – see below). Simple observation is seldom sufficient to determine how well the system meets the users' requirements since it does not always give insight into the their decision processes or attitude. Consequently users are asked to elaborate their actions by 'thinking aloud'. In this section we consider some of the techniques used to evaluate systems by observing user behavior.

Think aloud and cooperative evaluation

Think aloud is a form of observation where the user is asked to talk through what he is doing as he is being observed; for example, describing what he believes is happening, why he takes an action, what he is trying to do.

Think aloud has the advantage of simplicity; it requires little expertise to perform (though can be tricky to analyze fully) and can provide useful insight into problems with an interface. It can also be employed to observe how the system is actually used. It can be used for evaluation throughout the design process, using paper or simulated mock-ups for the earlier stages. However, the information provided is often subjective and may be selective, depending on the tasks provided. The process of observation can alter the way that people perform tasks and so provide a biased view. The very act of describing what you are doing often changes the way you do it – like the joke about the centipede who was asked how he walked . . .

A variation on think aloud is known as *cooperative evaluation* [240] in which the user is encouraged to see himself as a collaborator in the evaluation and not simply as an experimental participant. As well as asking the user to think aloud at the beginning of the session, the evaluator can ask the user questions (typically of the 'why?' or 'what-if?' type) if his behavior is unclear, and the user can ask the evaluator for clarification if a problem arises. This more relaxed view of the think aloud process has a number of advantages:

- the process is less constrained and therefore easier to learn to use by the evaluator
- the user is encouraged to criticize the system
- the evaluator can clarify points of confusion at the time they occur and so maximize the effectiveness of the approach for identifying problem areas.

The usefulness of think aloud, cooperative evaluation and observation in general is largely dependent on the effectiveness of the recording method and subsequent analysis. The record of an evaluation session of this type is known as a *protocol*, and there are a number of methods from which to choose.

Protocol analysis

Methods for recording user actions include the following:

Paper and pencil This is primitive, but cheap, and allows the analyst to note interpretations and extraneous events as they occur. However, it is hard to get detailed information, as it is limited by the analyst's writing speed. Coding schemes for frequent activities, developed during preliminary studies, can improve the rate of recording substantially, but can take some time to develop. A variation of paper and pencil is the use of a notebook computer for direct entry, but then one is limited to the analyst's typing speed, and one loses the flexibility of paper for writing styles, quick diagrams and spatial layout. If this is the only recording facility available then a specific note-taker, separate from the evaluator, is recommended.

Audio recording This is useful if the user is actively 'thinking aloud'. However, it may be difficult to record sufficient information to identify exact actions in later analysis, and it can be difficult to match an audio recording to some other form of protocol (such as a handwritten script).

Video recording This has the advantage that we can see *what* the participant is doing (*as long as* the participant stays within the range of the camera). Choosing suitable camera positions and viewing angles so that you get sufficient detail and yet keep the participant in view is difficult. Alternatively, one has to ask the participant not to move, which may not be appropriate for studying normal behavior! For single-user computer-based tasks, one typically uses two video cameras, one looking at the computer screen and one with a wider focus including the user's face and hands. The former camera may not be necessary if the computer system is being logged.

Computer logging It is relatively easy to get a system automatically to record user actions at a keystroke level, particularly if this facility has been considered early in the design. It can be more difficult with proprietary software where source code is not available (although some software now provides built-in logging and playback facilities). Obviously, computer logging only tells us what the user is doing on the system, but this may be sufficient for some purposes. Keystroke data are also 'semantics free' in that they only tell us about the lowest-level actions, not why they were performed or how they are structured (although slight pauses and gaps can give clues). Direct logging has the advantages that it is cheap (except in terms of disk storage), unobtrusive and can be used for *longitudinal studies*, where we look at one or more users over periods of weeks or months. Technical problems with it are that the sheer volume of data can become unmanageable without automatic analysis, and that one often has to be careful to restore the state of the system (file contents, etc.) before replaying the logs.

User notebooks The participants themselves can be asked to keep logs of activity/problems. This will obviously be at a very coarse level – at most, records every few minutes and, more likely, hourly or less. It also gives us 'interpreted' records, which have advantages and problems. The technique is especially useful in longitudinal studies, and also where we want a log of unusual or infrequent tasks and problems.

In practice, one uses a mixture of recording methods as they complement one another. For instance, we may keep a paper note of special events and circumstances, even when we have more sophisticated audio/visual recording. Similarly, we may use separate audio recording, even where a video recorder is used, as the quality of specialist audio recording is better than most built-in video microphones. In addition, we may use stereo audio recording, which helps us to locate out-of-screen noises. If one is using a collection of different sources, say audio, video (×2) and keystroke logging, there is considerable difficulty in synchronizing them during play-back. Most video recorders can superimpose an on-screen clock, which can help, but ideally one uses specialized equipment that can automatically synchronize the different sources, possibly merging several video displays onto a single screen. Unfortunately, this sort of equipment is often only available in specialized laboratories.

With both audio and video recording, a major problem is *transcription*. Typing a transcript from a tape is not the same as taped dictation. The conversation will typically consist of part or broken sentences, mumbled words and inarticulated noises. In addition, the transcript will need annotating with the different voices (which may only be clear from context) and with non-verbal items such as pauses, emphases, equipment noises, phones ringing, etc. A good audio-typist will be accustomed to completing mumbled words and correcting ungrammatical sentences – typing *exactly* what is recorded may prove difficult. Some practitioners say that the use of typists is not good practice anyway as the analyst will miss many nuances that are lost in the written transcript. However, if you wish to produce your own typed transcripts from tape, a course in touch-typing is highly recommended.

For video transcription, professional typists are not an option; there is no standard way of annotating video recordings, and the analyst must invent notations to suit the particular circumstances. The scale of this task is not to be underestimated. It is common to talk to practitioners who have tens or hundreds of hours of video recording, but have only analyzed tiny fragments in detail. Of course, the fragments will have been chosen after more extensive perusal of the material, but it certainly removes any idea of comprehensive coverage.

Coding can be introduced to indicate particular events but it is sometimes difficult to determine a suitable coding scheme and to use this consistently, particularly if more than one person is doing the coding. A range of transcribers should therefore test coding schemes to ensure that they are being interpreted appropriately for a particular data set.

Automatic protocol analysis tools

Analyzing protocols, whether video, audio or system logs, is time consuming and tedious by hand. It is made harder if there is more than one stream of data to synchronize. One solution to this problem is to provide automatic analysis tools to support the task. These offer a means of editing and annotating video, audio and system logs and synchronizing these for detailed analysis.

EVA (*Experimental Video Annotator*) is a system that runs on a multimedia workstation with a direct link to a video recorder [220]. The evaluator can devise a set of buttons indicating different events. These may include timestamps and snapshots, as well as notes of expected events and errors. The buttons are used within a recording session by the evaluator to annotate the video with notes. During the session the user works at a workstation and is recorded, using video and perhaps audio and system logging as well. The evaluator uses the multimedia workstation running EVA. On the screen is the live video record and a view of the user's screen (see Figure 9.4). The evaluator can use the buttons to tag interesting events as they occur and can record additional notes using a text editor. After the session, the evaluator can ask to review the tagged segments and can then use these and standard video controls to search the information. Links can be made with other types of record such as audio and system logs. A system such as EVA alleviates the burden of video analysis but it is not without its problems. The act of tagging and annotating events can prevent the evaluator from actually concentrating on the events themselves. This may mean that events are missed or tagged late.

Commercial systems such as Observer Pro from Noldus have similar functionality to EVA; portable versions are now available for use in field studies (www.noldus.com).

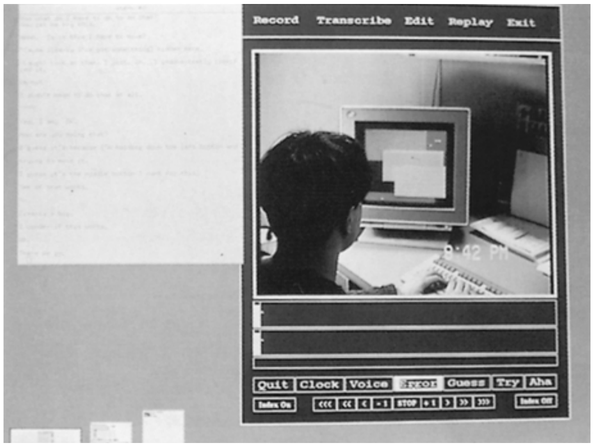


Figure 9.4 EVA: an automatic protocol analysis tool. Source: Wendy Mackay

The *Workplace project* at Xerox PARC [348] also includes a system to aid protocol analysis. The main emphasis here is to support the analysis of synchronized information from different data streams, such as video, audio, notes and diagrams. Each data stream is viewed in an aligned display so that it is possible to compare the records of each for a given point in the interaction. The alignment may be based on timestamps or on an event or action and is implemented using hypertext links.

A third example is DRUM [223], which also provides video annotation and tagging facilities. DRUM is part of the MUSiC (Measuring the Usability of Systems in Context/Metrics for Usability Standards in Computing) toolkit, which supports a complete methodology for evaluation, based upon the application of usability metrics on analytic metrics, cognitive workload, performance and user satisfaction. DRUM is concerned particularly with measuring performance. The methodology provides a range of tools as well as DRUM, including manuals, questionnaires, analysis software and databases.

Systems such as these are extremely important as evaluation tools since they offer a means of handling the data that are collected in observational studies and allowing a more systematic approach to the analysis. The evaluator's task is facilitated and it is likely that more valuable observations will emerge as a result.

Post-task walkthroughs

Often data obtained via direct observation lack interpretation. We have the basic actions that were performed, but little knowledge as to why. Even where the participant has been encouraged to think aloud through the task, the information may be at the wrong level. For example, the participant may say 'and now I'm selecting the undo menu', but not tell us what was wrong to make undo necessary. In addition, a think aloud does not include information such as alternative, but not pursued, actions.

A walkthrough attempts to alleviate these problems, by reflecting the participants' actions back to them after the event. The transcript, whether written or recorded, is replayed to the participant who is invited to comment, or is directly questioned by the analyst. This may be done straightaway, when the participant may actually remember why certain actions were performed, or after an interval, when the answers are more likely to be the participant's post hoc interpretation. (In fact, interpretation is likely even in the former case.) The advantage of a delayed walkthrough is that the analyst has had time to frame suitable questions and focus on specific incidents. The disadvantage is a loss of freshness.

There are some circumstances when the participant cannot be expected to talk during the actual observation, for instance during a critical task, or when the task is too intensive. In these circumstances, the post-task walkthrough is the only way to obtain a subjective viewpoint on the user's behavior. There is also an argument that it is preferable to minimize non-task-related talk during direct observation in order to get as natural a performance as possible. Again this makes the walkthrough essential.

9.4.4 Query techniques

Another set of evaluation techniques relies on asking the user about the interface directly. Query techniques can be useful in eliciting detail of the user's view of a system. They embody the philosophy that states that the best way to find out how a system meets user requirements is to 'ask the user'. They can be used in evaluation and more widely to collect information about user requirements and tasks. The advantage of such methods is that they get the user's viewpoint directly and may reveal issues that have not been considered by the designer. In addition, they are relatively simple and cheap to administer. However, the information gained is necessarily subjective, and may be a 'rationalized' account of events rather than a wholly accurate one. Also, it may be difficult to get accurate feedback about alternative designs if the user has not experienced them, which limits the scope of the information that can be gleaned. However, the methods provide useful supplementary material to other methods. There are two main types of query technique: interviews and questionnaires.

Interviews

Interviewing users about their experience with an interactive system provides a direct and structured way of gathering information. Interviews have the advantages that the level of questioning can be varied to suit the context and that the evaluator can probe the user more deeply on interesting issues as they arise. An interview will usually follow a top-down approach, starting with a general question about a task and progressing to more leading questions (often of the form 'why?' or 'what if?') to elaborate aspects of the user's response.

Interviews can be effective for high-level evaluation, particularly in eliciting information about user preferences, impressions and attitudes. They may also reveal problems that have not been anticipated by the designer or that have not occurred under observation. When used in conjunction with observation they are a useful means of clarifying an event (compare the post-task walkthrough).

In order to be as effective as possible, the interview should be planned in advance, with a set of central questions prepared. Each interview is then structured around these questions. This helps to focus the purpose of the interview, which may, for instance, be to probe a particular aspect of the interaction. It also helps to ensure a base of consistency between the interviews of different users. That said, the evaluator may, of course, choose to adapt the interview form to each user in order to get the most benefit: the interview is not intended to be a controlled experimental technique.

Questionnaires

An alternative method of querying the user is to administer a questionnaire. This is clearly less flexible than the interview technique, since questions are fixed in advance, and it is likely that the questions will be less probing. However, it can be used to reach a wider participant group, it takes less time to administer, and it can be analyzed more rigorously. It can also be administered at various points in the design process, including during requirements capture, task analysis and evaluation, in order to get information on the user's needs, preferences and experience.

Given that the evaluator is not likely to be directly involved in the completion of the questionnaire, it is vital that it is well designed. The first thing that the evaluator must establish is the purpose of the questionnaire: what information is sought? It is also useful to decide at this stage how the questionnaire responses are to be analyzed. For example, do you want specific, measurable feedback on particular interface features, or do you want the user's impression of using the interface?

There are a number of styles of question that can be included in the questionnaire. These include the following:

General These are questions that help to establish the background of the user and his place within the user population. They include questions about age, sex, occupation, place of residence, and so on. They may also include questions on previous experience with computers, which may be phrased as open-ended, multi-choice or scalar questions (see below).

Open-ended These ask the user to provide his own unprompted opinion on a question, for example 'Can you suggest any improvements to the interface?'. They are useful for gathering general subjective information but are difficult to analyze in any rigorous way, or to compare, and can only be viewed as supplementary. They are also most likely to be missed out by time-conscious respondents! However, they may identify errors or make suggestions that have not been considered by the designer. A special case of this type is where the user is asked for factual information, for example how many commands were used.

Scalar These ask the user to judge a specific statement on a numeric scale, usually corresponding to a measure of agreement or disagreement with the statement. For example,

It is easy to recover from mistakes.
Disagree 1 2 3 4 5 Agree

The granularity of the scale varies: a coarse scale (say, from 1 to 3) gives a clear indication of the meaning of the numbers (disagree, neutral and agree). However, it gives no room for varying levels of agreement, and users may therefore be tempted to give neutral responses to statements that they do not feel strongly about but with which they mildly disagree or agree. A very fine scale (say 1 to 10) suffers from the opposite problem: the numbers become difficult to interpret in a consistent way. One user will undoubtedly interpret the scale differently from another. A middle ground is therefore advisable. Scales of 1 to 5 or 1 to 7 have been used effectively. They are fine enough to allow users to differentiate adequately but still retain clarity in meaning. It can help to provide an indication of the meaning of intermediate scalar values. Odd-numbered scales are used most often but it is possible to use even-numbered scales (e.g. 1–6) if the 'neutral' option is not wanted. This does not allow for fence sitting – except decisively by selecting $3\frac{1}{2}$!).

Multi-choice Here the respondent is offered a choice of explicit responses, and may be asked to select only one of these, or as many as apply. For example,

How do you most often get help with the system (tick one)?

Online manual	❑
Contextual help system	❑
Command prompt	❑
Ask a colleague	❑

Which types of software have you used (tick all that apply)?

Word processor	❑
Database	❑
Spreadsheet	❑
Expert system	❑
Online help system	❑
Compiler	❑

These are particularly useful for gathering information on a user's previous experience. A special case of this type is where the offered choices are 'yes' or 'no'.

Ranked These place an ordering on items in a list and are useful to indicate a user's preferences. For example,

Please rank the usefulness of these methods of issuing a command (1 most useful, 2 next, 0 if not used).

Menu selection	❑
Command line	❑
Control key accelerator	❑

These question types are all useful for different purposes, as we have noted. However, in order to reduce the burden of effort on the respondent, and so encourage a high response rate amongst users, it is best to use closed questions, such as scalar, ranked or multi-choice, as much as possible. These provide the user with alternative responses and so reduce the effort required. They also have the advantage of being easier to analyze. Responses can be analyzed in a number of ways, from determining simple percentages for each response, to looking at correlations and factor analysis. For more detail on available methods the reader is referred to the recommended reading list at the end of the chapter.

Whatever type of questionnaire is planned, it is wise to carry out a pilot study. This allows any problems with the questionnaire design to be ironed out before the questionnaire is distributed to potentially hundreds of users! The questionnaire should be tested on four or five users to see if the questions are comprehensible and the results are as expected and can be used in the manner intended. If users seem to be misunderstanding a particular question, it can then be rephrased (and retested) before the final version is sent out.

Distribution of questionnaires can also be problematic. It is important that the respondents are representative of the user population but you also need to ensure that you are able to reach as many potential respondents as possible. Return rate for questionnaires is quite low (often 25–30%) so many more need to be sent out to get a reasonable return. Questionnaires should ideally be distributed to a random subset of the user population. So, for example, if the population is all workers in a company, one may choose to send a questionnaire to every fourth person on an alphabetically ordered personnel list. However, questionnaires are now often distributed via the internet, either by email, where potential respondents can be selected randomly, or via a website, where the respondents are limited to those who visit the site and who may not be representative. In practice, questionnaire respondents are self-selecting anyway, in that only those who choose to respond are included in the study; if the questionnaire is designed to capture demographic information about each respondent then the level of representativeness (or otherwise) can be determined from the responses.

Worked exercise *You have been asked to compare user performance and preferences with two different learning systems, one using hypermedia (see Chapter 21), the other sequential lessons. Design a questionnaire to find out what the users think of the system. How would you go about comparing user performance with these two systems?*

Answer Assume that all users have used both systems.

Questionnaire
Consider the following questions in designing the questionnaire:

- what information is required?
- how is the questionnaire to be analyzed?

You are particularly interested in user preferences so questions should focus on different aspects of the systems and try to measure levels of satisfaction. The use of scales will make responses for each system easier to compare.

Table 9.3 shows an example questionnaire.

To test performance you would design an experiment where two groups of participants learn the same material using the two systems, and test how well they have learned (using a standard measurable test).

Participants User group

IV (Independent Variable) Style of learning system

DV (Dependent Variable) Performance (measured as test score)

Design Between-subjects design

Table 9.3 Questionnaire to compare two systems

PART I: Repeat for each system

Indicate your agreement or disagreement with the following statements. (1 indicates complete disagreement and 5 complete agreement.)

The system tells me what to do at every point.	Disagree 1 2 3 4 5 Agree
It is easy to recover from mistakes.	Disagree 1 2 3 4 5 Agree
It is easy to get help when needed.	Disagree 1 2 3 4 5 Agree
I always know what the system is doing.	Disagree 1 2 3 4 5 Agree
I always know where I am in the training material.	Disagree 1 2 3 4 5 Agree
I have learned the material well using the system.	Disagree 1 2 3 4 5 Agree
I could have learned the material more effectively using a book.	Disagree 1 2 3 4 5 Agree
I always know how well I am doing.	Disagree 1 2 3 4 5 Agree

PART II: Comparing both systems:

Which system (choose 1) was most:

Helpful to use	A	B
Efficient to use	A	B
Enjoyable to use	A	B

Please add any comments you have about either system:

9.4.5 Evaluation through monitoring physiological responses

One of the problems with most evaluation techniques is that we are reliant on observation and the users telling us what they are doing and how they are feeling. What if we were able to measure these things directly? Interest has grown recently in the use of what is sometimes called objective usability testing, ways of monitoring physiological aspects of computer use. Potentially this will allow us not only to see more clearly exactly what users do when they interact with computers, but also to measure how they feel. The two areas receiving the most attention to date are eye tracking and physiological measurement.

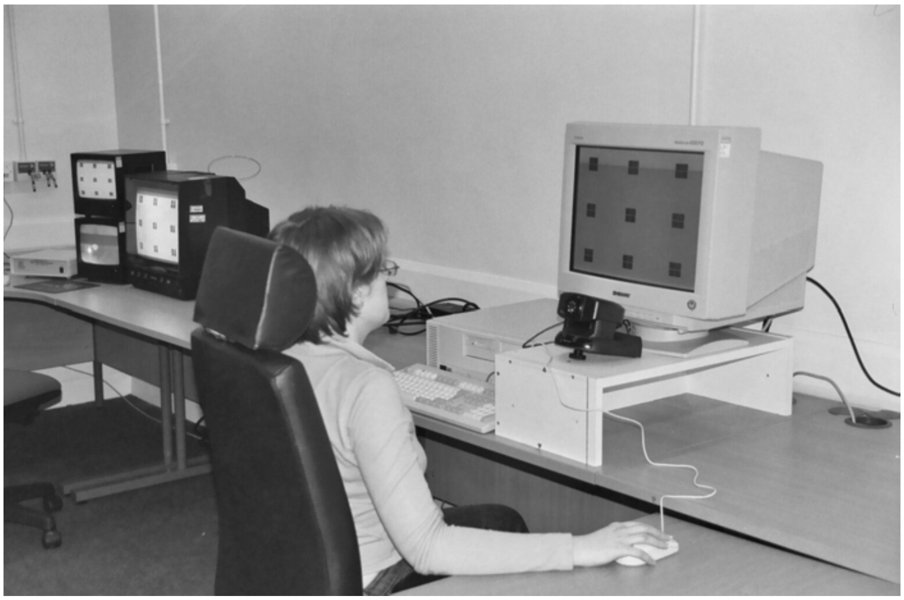


Figure 9.5 Eye-tracking equipment. Source: Courtesy of J. A. Renshaw

Eye tracking for usability evaluation

Eye tracking has been possible for many years, but recent improvements in hardware and software have made it more viable as an approach to measuring usability. The original eye trackers required highly invasive procedures where eye caps were attached to the cornea under anaesthetic. Clearly inappropriate for usability testing! Modern systems vary: some use a head-mounted camera to monitor the eye, but the most sophisticated do not involve any contact between the equipment and the participant, with the camera and light sources mounted in desk units (see Figures 9.5, 9.6) [112].

Furthermore, there have been rapid improvements in the software available both for the control of eye-tracking equipment and the analysis and visualization of the large volumes of data it produces.

Eye movements are believed to reflect the amount of cognitive processing a display requires and, therefore, how easy or difficult it is to process [150]. So measuring not only where people look, but also their patterns of eye movement, may tell us which areas of a screen they are finding easy or difficult to understand. Eye movement measurements are based on fixations, where the eye retains a stable position for a period of time, and saccades, where there is rapid ballistic eye movement from one point of interest to another. There are many possible measurements related to usability evaluation including:

Number of fixations The more fixations the less efficient the search strategy.

Fixation duration Longer fixations may indicate difficulty with a display.

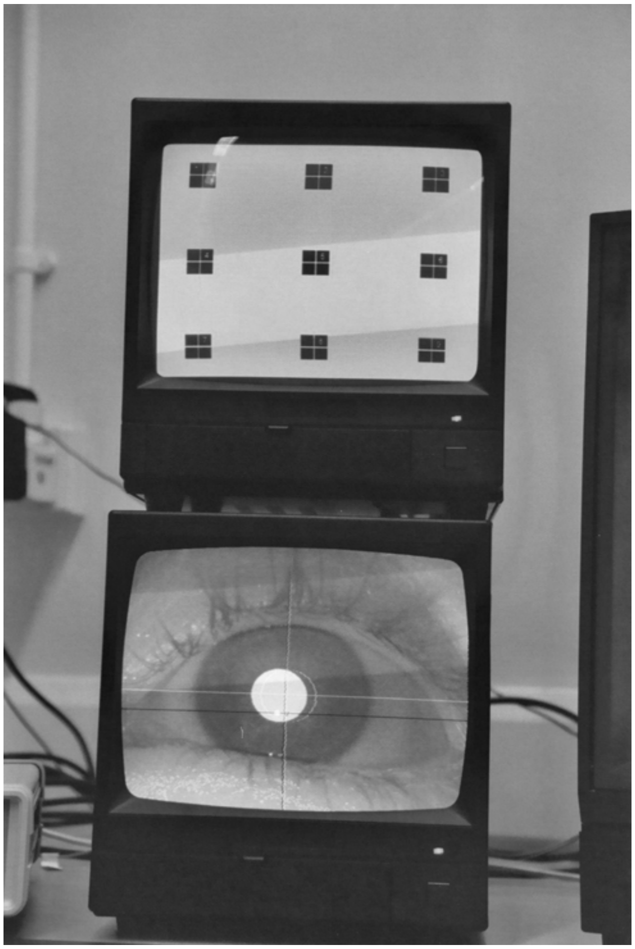


Figure 9.6 Calibrating the eye tracker. Source: Courtesy of J. A. Renshaw

Scan path indicating areas of interest, search strategy and cognitive load. Moving straight to a target with a short fixation at the target is the optimal scan path but plotting scan paths and fixations can indicate what people look at, how often and for how long.

Eye tracking for usability is still very new and equipment is prohibitively expensive for everyday use. However, it is a promising technique for providing insights into what really attracts the eye in website design and where problem areas are in system use. More research is needed to interpret accurately the meaning of the various eye movement measurements, as well as to develop more accessible and robust equipment. But, given the potential for gathering new data measurements relatively unobtrusively, it is likely that eye tracking will become part of the standard equipment for usability laboratories in the coming few years.

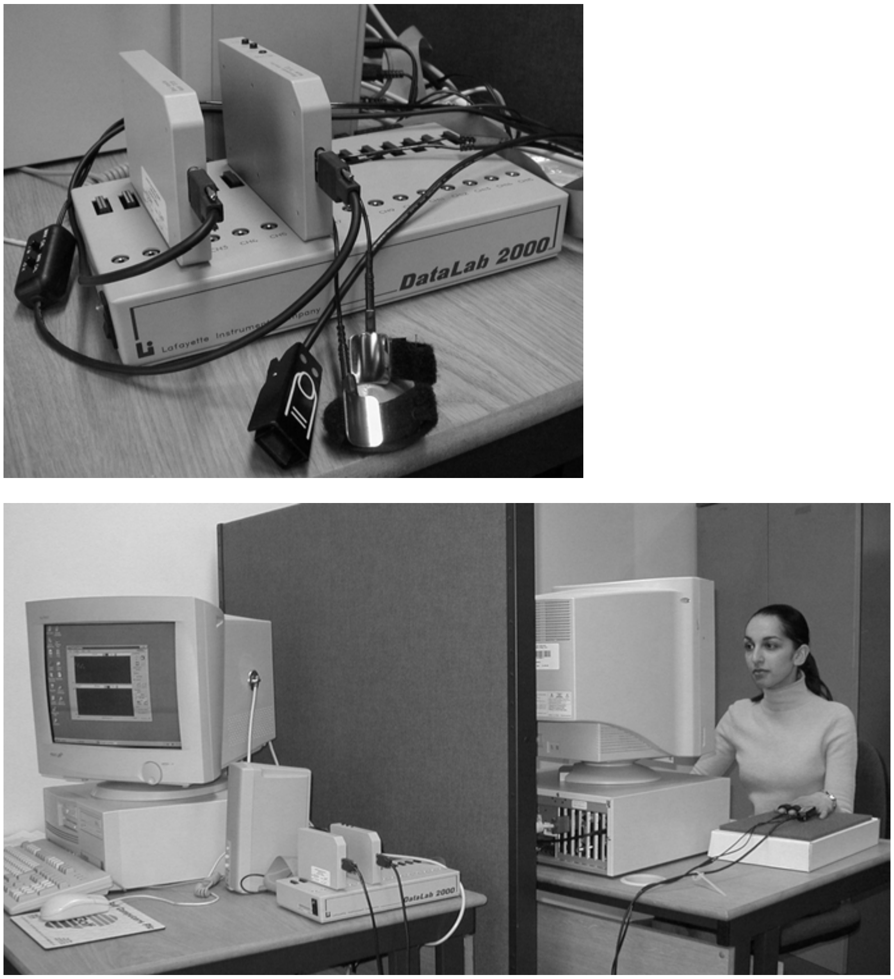


Figure 9.7 Data Lab Psychophysiology equipment showing some of the sensors (above) and a typical experimental arrangement (below) with sensors attached to the participant's fingers and the monitoring software displayed on the evaluator's machine. Source: Courtesy of Dr R. D. Ward

Physiological measurements

As we saw in Chapter 1, emotional response is closely tied to physiological changes. These include changes in heart rate, breathing and skin secretions. Measuring these physiological responses may therefore be useful in determining a user's emotional response to an interface [288, 363]. Could we determine which interaction events really cause a user stress or which promote relaxation?

Physiological measurement involves attaching various probes and sensors to the user (see Figure 9.7). These measure a number of factors:

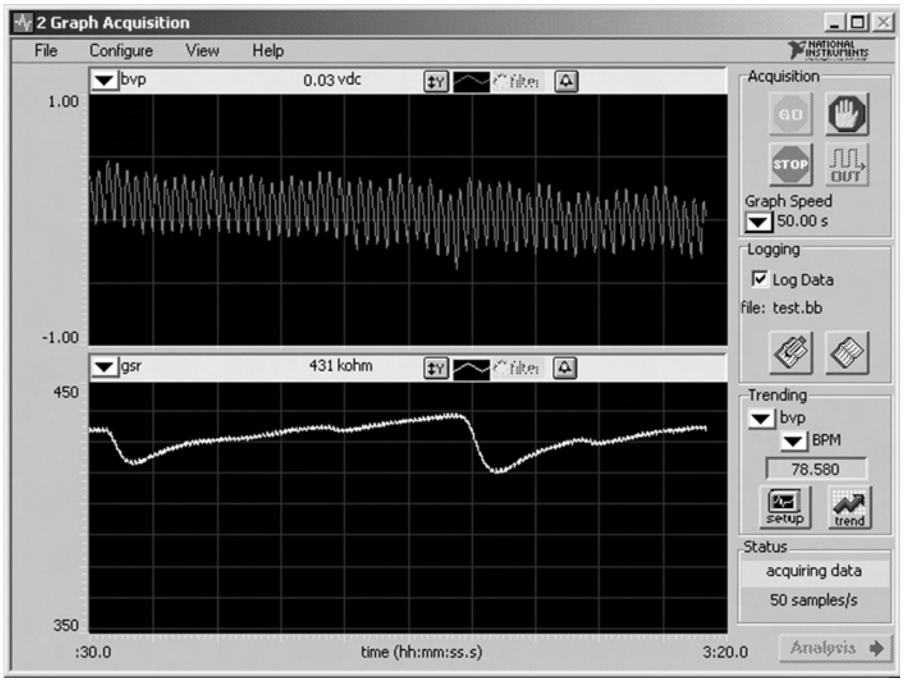


Figure 9.8 Output of monitoring pulse rate (above) and skin conductivity (below). Source: Screen shot courtesy of Dr R. D. Ward; frame source: National Instruments BioBench software

Heart activity, indicated by blood pressure, volume and pulse. These may respond to stress or anger.

Activity of the sweat glands, indicated by skin resistance or galvanic skin response (GSR). These are thought to indicate levels of arousal and mental effort.

Electrical activity in muscle, measured by the electromyogram (EMG). These appear to reflect involvement in a task.

Electrical activity in the brain, measured by the electroencephalogram (EEG). These are associated with decision making, attention and motivation.

Figure 9.8 illustrates the output obtained from such measurements.

One of the problems with applying these measurements to interaction events is that it is not clear what the relationship between these events and measurements might be. For example, if increased pulse rate is observed during an interactive task, does that indicate frustration with the interface or stress at being unable to complete the task? How will physiological changes differ in response to discrete events or to continuous interface use? Is it possible to map patterns of physiological measurement to specific emotional states?

These are still research questions but the approach is interesting, as it offers a potential means of objectively capturing information about the user's emotional state, which, as we saw in Chapter 1, impacts on our ability to solve problems and perform tasks.

9.5 CHOOSING AN EVALUATION METHOD

As we have seen in this chapter, a range of techniques is available for evaluating an interactive system at all stages in the design process. So how do we decide which methods are most appropriate for our needs? There are no hard and fast rules in this – each method has its particular strengths and weaknesses and each is useful if applied appropriately. However, there are a number of factors that should be taken into account when selecting evaluation techniques. These also provide a way of categorizing the different methods so that we can compare and choose between them. In this final section we will consider these factors.

9.5.1 Factors distinguishing evaluation techniques

We can identify at least eight factors that distinguish different evaluation techniques and therefore help us to make an appropriate choice. These are:

- the stage in the cycle at which the evaluation is carried out
- the style of evaluation
- the level of subjectivity or objectivity of the technique
- the type of measures provided
- the information provided
- the immediacy of the response
- the level of interference implied
- the resources required.

Design vs. implementation

The first factor to affect our choice of evaluation method is the stage in the design process at which evaluation is required. As we saw earlier in this chapter, it is desirable to include evaluation of some sort throughout the design process. The main distinction between evaluation of a design and evaluation of an implementation is that in the latter case a physical artifact exists. This may be anything from a paper mock-up to a full implementation, but it is something concrete that can be tested. Evaluation of a design, on the other hand, precedes this stage and seeks instead to provide information to feed the development of the physical artifact.

Roughly speaking, evaluation at the design stage needs to be quick and cheap so might involve design experts only and be analytic, whereas evaluation of the implementation needs to be more comprehensive, so brings in users as participants. There are, of course, exceptions to this: participatory design (see Chapter 13) involves users throughout the design process, and techniques such as cognitive walk-through are expert based and analytic but can be used to evaluate implementations as well as designs.

Early evaluation, whether of a design or an early prototype or mock-up, will bring the greatest pay-off since problems can be easily resolved at this stage. As more commitment is made to a particular design in the implementation, it becomes increasingly difficult for changes to be made, no matter what the evaluation suggests. Ironically, the most resources are often ploughed into late evaluations. This is less profitable and should be avoided, although obviously some evaluation with users is required with a complete, or almost complete, system, since some elements (such as system performance) will only be evident in the working system.

Laboratory vs. field studies

We have already discussed the pros and cons of these two styles of evaluation. Laboratory studies allow controlled experimentation and observation while losing something of the naturalness of the user's environment. Field studies retain the latter but do not allow control over user activity. Ideally the design process should include both styles of evaluation, probably with laboratory studies dominating the early stages and field studies conducted with the new implementation.

Subjective vs. objective

Evaluation techniques also vary according to their objectivity – some techniques rely heavily on the interpretation of the evaluator, others would provide similar information for anyone correctly carrying out the procedure. The more subjective techniques, such as cognitive walkthrough or think aloud, rely to a large extent on the knowledge and expertise of the evaluator, who must recognize problems and understand what the user is doing. They can be powerful if used correctly and will provide information that may not be available from more objective methods. However, the problem of evaluator bias should be recognized and avoided. One way to decrease the possibility of bias is to use more than one evaluator. Objective techniques, on the other hand, should produce repeatable results, which are not dependent on the persuasion of the particular evaluator. Controlled experiments are an example of an objective measure. These avoid bias and provide comparable results but may not reveal the unexpected problem or give detailed feedback on user experience. Ideally, both objective and subjective approaches should be used.

The extent to which the results are dependent on the subjective response of the user also varies. At one extreme is asking for the user's opinions, which is highly subjective; at the other is measuring physiological changes in the body, which are outside the user's control.

Qualitative vs. quantitative measures

The type of measurement provided by the evaluation technique is also an important consideration. There are two main types: *quantitative measurement* and *qualitative* *measurement*. The former is usually numeric and can be easily analyzed using statistical techniques. The latter is non-numeric and is therefore more difficult to analyze, but can provide important detail that cannot be determined from numbers. The type of measure is related to the subjectivity or objectivity of the technique, with subjective techniques tending to provide qualitative measures and objective techniques, quantitative measures. This is not a hard and fast rule, however. It is sometimes possible to quantify what is, in fact, qualitative information by mapping it onto a scale or similar measure. A common example of this is in questionnaires where qualitative information is being sought (for example, user preferences) but a quantitative scale is used. This is also common in experimental design where factors such as the quality of the user's performance are used as dependent variables, and measured on a quantitative scale.

Information provided

The level of information required from an evaluation may also vary. The information required by an evaluator at any stage of the design process may range from low-level information to enable a design decision to be made (for example, which font is most readable) to higher-level information, such as 'Is the system usable?'. Some evaluation techniques, such as controlled experiments, are excellent at providing low-level information – an experiment can be designed to measure a particular aspect of the interface. Higher-level information can be gathered using questionnaire and interview techniques, which provide a more general impression of the user's view of the system.

Immediacy of response

Another factor distinguishing evaluation techniques is the immediacy of the response they provide. Some methods, such as think aloud, record the user's behavior at the time of the interaction itself. Others, such as post-task walkthrough, rely on the user's recollection of events. Such recollection is liable to suffer from bias in recall and reconstruction, with users interpreting events according to their preconceptions. Recall may also be incomplete. However, immediate techniques can also be problematic, since the process of measurement can actually alter the way the user works.

Intrusiveness

Related to the immediacy of the response is the intrusiveness of the technique itself. Certain techniques, particularly those that produce immediate measurements, are obvious to the user during the interaction and therefore run the risk of influencing the way the user behaves. Sensitive activity on the part of the evaluator can help to reduce this but cannot remove it altogether. Most immediate evaluation techniques are intrusive, with the exception of automatic system logging. Unfortunately, this is limited in the information that it can provide and is difficult to interpret.

Resources

The final consideration when selecting an evaluation technique is the availability of resources. Resources to consider include equipment, time, money, participants, expertise of evaluator and context. Some decisions are forced by resource limitations: it is not possible to produce a video protocol without access to a video camera (and probably editing facilities as well). However, other decisions are not so clear cut. For example, time and money may be limited, forcing a choice between two possible evaluations. In these circumstances, the evaluator must decide which evaluation tactic will produce the most effective and useful information for the system under consideration. It may be possible to use results from other people's experiments to avoid having to conduct new experiments.

Some techniques are more reliant on evaluator expertise than others, for example the more formal analytic techniques. If evaluator expertise is limited, it may be more practical to use more simple heuristic methods than methods that require understanding of user goal structures and so on.

Finally, the context in which evaluation can occur will influence what can be done. For practical reasons it may not be possible to gain access to the intended users of a system (if it is a general system, for example). Or it may not be feasible to test the system in its intended environment (for example, a system for a space station or a defence system). In these circumstances simulations must be used.

9.5.2 A classification of evaluation techniques

Using the factors discussed in the previous section we can classify the evaluation techniques we have considered in this chapter. This allows us to identify the techniques that most closely fit our requirements. Table 9.4 shows the classification for

Table 9.4 Classification of analytic evaluation techniques

	Cognitive walkthrough	Heuristic evaluation	Review based	Model based
Stage	Throughout	Throughout	Design	Design
Style	Laboratory	Laboratory	Laboratory	Laboratory
Objective?	No	No	As source	No
Measure	Qualitative	Qualitative	As source	Qualitative
Information	Low level	High level	As source	Low level
Immediacy	N/A	N/A	As source	N/A
Intrusive?	No	No	No	No
Time	Medium	Low	Low–medium	Medium
Equipment	Low	Low	Low	Low
Expertise	High	Medium	Low	High

Table 9.5 Classification of experimental and query evaluation techniques

	Experiment	Interviews	Questionnaire
Stage	Throughout	Throughout	Throughout
Style	Laboratory	Lab/field	Lab/field
Objective?	Yes	No	No
Measure	Quantitative	Qualitative/ quantitative	Qualitative/ quantitative
Information	Low/high level	High level	High level
Immediacy	Yes	No	No
Intrusive?	Yes	No	No
Time	High	Low	Low
Equipment	Medium	Low	Low
Expertise	Medium	Low	Low

Table 9.6 Classification of observational evaluation techniques

	Think aloud[1]	Protocol analysis[2]	Post-task walkthrough
Stage	Implementation	Implementation	Implementation
Style	Lab/field	Lab/field	Lab/field
Objective?	No	No	No
Measure	Qualitative	Qualitative	Qualitative
Information	High/low level	High/low level	High/low level
Immediacy	Yes	Yes	No
Intrusive?	Yes	Yes[3]	No
Time	High	High	Medium
Equipment	Low	High	Low
Expertise	Medium	High	Medium

1 Assuming a simple paper and pencil record
2 Including video, audio and system recording
3 Except system logs

analytic techniques, Table 9.5 for experimental and query techniques, Table 9.6 for observational and Table 9.7 for monitoring techniques.

The classification is intended as a rough guide only – some of the techniques do not fit easily into such a classification since their use can vary considerably.

Table 9.7 Classification of monitoring evaluation techniques

	Eye tracking	Physiological measurement
Stage	Implementation	Implementation
Style	Lab	Lab
Objective?	Yes	Yes
Measure	Quantitative	Quantitative
Information	Low level	Low level
Immediacy	Yes	Yes
Intrusive?	No[1]	Yes
Time	Medium/high	Medium/high
Equipment	High	High
Expertise	High	High

1 If the equipment is not head mounted

9.6 SUMMARY

Evaluation is an integral part of the design process and should take place throughout the design life cycle. Its aim is to test the functionality and usability of the design and to identify and rectify any problems. It can also try to determine the user's attitude and response to the system. It can take place in a specialist laboratory or in the user's workplace, and may or may not involve active participation on the part of the user.

A design can be evaluated before any implementation work has started, to minimize the cost of early design errors. Most techniques for evaluation at this stage are analytic and involve using an expert to assess the design against cognitive and usability principles. Previous experimental results and modeling approaches can also provide insight at this stage. Once an artifact has been developed (whether a prototype or full system), experimental and observational techniques can be used to get both quantitative and qualitative results. Query techniques provide subjective information from the user. If more objective information is required, physiological monitoring can capture the user's physical responses to the system.

The choice of evaluation method is largely dependent on what is required of the evaluation. Evaluation methods vary in the stage at which they are commonly used and where they can be used. Some are more subjective than others and provide qualitative rather than quantitative measures. Some provide immediate information while others get feedback after the event. However, the more immediate methods also tend to intrude most seriously on the interaction. Finally, some require more resources in terms of time, equipment and expertise than others.

EXERCISES

9.1 In groups or pairs, use the cognitive walkthrough example, and what you know about user psychology (see Chapter 1), to discuss the design of a computer application of your choice (for example, a word processor or a drawing package). (**Hint:** Focus your discussion on one or two specific tasks within the application.)

9.2 What are the benefits and problems of using video in experimentation? If you have access to a video recorder, attempt to transcribe a piece of action and conversation (it does not have to be an experiment – a soap opera will do!). What problems did you encounter?

9.3 In Section 9.4.2, we saw that the observed results could be the result of interference. Can you think of alternative designs that may make this less likely? Remember that individual variation was very high, so you *must* retain a within-subjects design, but you may perform more tests on each participant.

9.4 Choose an appropriate evaluation method for each of the following situations. In each case identify:

(i) the participants
(ii) the technique used
(iii) representative tasks to be examined
(iv) measurements that would be appropriate
(v) an outline plan for carrying out the evaluation.

(a) You are at an early stage in the design of a spreadsheet package and you wish to test what type of icons will be easiest to learn.
(b) You have a prototype for a theatre booking system to be used by potential theatre-goers to reduce queues at the box office.
(c) You have designed and implemented a new game system and want to evaluate it before release.
(d) You have developed a group decision support system for a solicitor's office.
(e) You have been asked to develop a system to store and manage student exam results and would like to test two different designs prior to implementation or prototyping.

9.5 Complete the cognitive walkthrough example for the video remote control design.

9.6 In defining an experimental study, describe:

(a) how you as an experimenter would formulate the hypothesis to be supported or refuted by your study;
(b) how you would decide between a within-groups or between-groups experimental design with your subjects.

9.7 What are the factors governing the choice of an appropriate evaluation method for different interactive systems? Give brief details.

RECOMMENDED READING

C. Robson, *Experiment, Design and Statistics in Psychology*, 3rd edition, Penguin, 1994.
An accessible introduction to statistics and experimental design and analysis for the uninitiated, using worked examples throughout.

A. Monk, P. Wright, J. Haber and L. Davenport, *Improving your Human Computer Interface: A Practical Approach*, Prentice Hall, 1992.
An evaluator's guide to using the cooperative evaluation approach successfully.

M. Helander, editor, *Handbook of Human–Computer Interaction, Part V: Tools for Design and Evaluation*, North-Holland, 1988.
Reviews the major evaluation techniques.

C. Wharton, J. Rieman, C. Lewis and P. Polson, The cognitive walkthrough: a practitioner's guide. In J. Nielsen and R. L. Mack, editors, *Usability Inspection Methods*, John Wiley, 1994.
Describes the revised version of the cognitive walkthrough method of evaluation.

J. Nielsen, Heuristic evaluation. In J. Nielsen and R. L. Mack, editors, *Usability Inspection Methods*, John Wiley, 1994.
Covers the heuristic evaluation method in detail.

Interacting with Computers, Vol. 14, No. 2, special issue on emotion in interaction, February 2002.
A special journal issue containing several papers on the use of psychophysiology in studying emotional response to interaction.

A. T. Duchowski, *Eye Tracking Methodology: Theory and Practice*, Springer-Verlag, 2003.
A recent book providing a detailed introduction to eye-tracking theory, equipment and applications.

10 Universal design

OVERVIEW

- Universal design is about designing systems so that they can be used by anyone in any circumstance.
- Multi-modal systems are those that use more than one human input channel in the interaction.
- These systems may, for example, use:
  - speech
  - non-speech sound
  - touch
  - handwriting
  - gestures.
- Universal design means designing for diversity, including:
  - people with sensory, physical or cognitive impairment
  - people of different ages
  - people from different cultures and backgrounds.

10.1 INTRODUCTION

We have already discussed the importance of designing for the user, considering human abilities and requirements. But is it possible to generalize about people and, if not, how do we address the issue of human diversity in our designs?

The discussion that we had on human psychology in Chapter 1 talked about general human abilities and, in reality, people are much more varied than the discussion suggests. People have different abilities and weaknesses; they come from different backgrounds and cultures; they have different interests, viewpoints and experiences; they are different ages and sizes. All of these things have an impact on the way in which an individual will use a particular computing application and, indeed, on whether or not they can use it at all. Given such diversity, we cannot assume a 'typical' user or design only for people like ourselves.

Universal design is the process of designing products so that they can be used by as many people as possible in as many situations as possible. In our case, this means particularly designing interactive systems that are usable by anyone, with any range of abilities, using any technology platform. This can be achieved by designing systems either to have built in redundancy or to be compatible with assistive technologies. An example of the former might be an interface that has both visual and audio access to commands; an example of the latter, a website that provides text alternatives for graphics, so that it can be read using a screen reader.

In this chapter, we will look at universal design in more detail. We will begin by examining seven principles of universal design. We will then look at multi-modal technology and how it can help to provide redundancy in interaction. Having identified some of the available technologies at our disposal, we will look in more detail at the particular areas of human diversity that we need to address.

10.2 UNIVERSAL DESIGN PRINCIPLES

We have defined universal design as 'the process of designing products so that they can be used by as many people as possible in as many situations as possible'. But what does that mean in practice? Is it possible to design anything so that anyone can use it – and if we could, how practical would it be? Wouldn't the cost be prohibitive? In reality, we may not be able to design everything to be accessible to everyone, and we certainly cannot ensure that everyone has the same experience of using a product, but we can work toward the aim of universal design and try to provide an *equivalent* experience.

Although it may seem like a huge task, universal design does not have to be complex or costly. In fact, if you are observant, you will see many examples of design that attempt to take account of user diversity. Next time you cross the road, look at the pavement. The curb may be lowered, to enable people who use wheelchairs to cross more easily. The paving near the curb may be of a different texture – with raised bumps or ridges – to enable people who cannot see to find the crossing point. Notice how many modern buildings have automatic doors that open on approach. Or lifts that offer both visual and auditory notification of the floor reached. And, whilst these designs make the crossing, the building and the lift more accessible to people who have disabilities, notice too how they also help other users. The parent with a child in a buggy, or the traveller with wheeled luggage, can cross the road more easily; the shopper with heavy bags, or the small child, can enter the building; and people are less likely to miss their floor because they weren't paying attention. Universal design is primarily about trying to ensure that you do not exclude anyone through the design choices you make but, by giving thought to these issues, you will invariably make your design better for everyone.

In the late 1990s a group at North Carolina State University in the USA proposed seven general principles of universal design [333]. These were intended to cover all areas of design and are equally applicable to the design of interactive systems. These principles give us a framework in which to develop universal designs.

Principle one is *equitable use*: the design is useful to people with a range of abilities and appealing to all. No user is excluded or stigmatized. Wherever possible, access should be the same for all; where identical use is not possible, equivalent use should be supported. Where appropriate, security, privacy and safety provision should be available to all.

Principle two is *flexibility in use*: the design allows for a range of ability and preference, through choice of methods of use and adaptivity to the user's pace, precision and custom.

Principle three is that the system be *simple and intuitive to use*, regardless of the knowledge, experience, language or level of concentration of the user. The design needs to support the user's expectations and accommodate different language and literacy skills. It should not be unnecessarily complex and should be organized to facilitate access to the most important areas. It should provide prompting and feedback as far as possible.

Principle four is *perceptible information*: the design should provide effective communication of information regardless of the environmental conditions or the user's abilities. Redundancy of presentation is important: information should be represented in different forms or modes (e.g. graphic, verbal, text, touch). Essential information should be emphasized and differentiated clearly from the peripheral content. Presentation should support the range of devices and techniques used to access information by people with different sensory abilities.

Principle five is *tolerance for error*: minimizing the impact and damage caused by mistakes or unintended behavior. Potentially dangerous situations should be removed or made hard to reach. Potential hazards should be shielded by warnings. Systems should fail safe from the user's perspective and users should be supported in tasks that require concentration.

Principle six is *low physical effort*: systems should be designed to be comfortable to use, minimizing physical effort and fatigue. The physical design of the system should allow the user to maintain a natural posture with reasonable operating effort. Repetitive or sustained actions should be avoided.

Principle seven requires *size and space for approach and use*: the placement of the system should be such that it can be reached and used by any user regardless of body size, posture or mobility. Important elements should be on the line of sight for both seated and standing users. All physical components should be comfortably reachable by seated or standing users. Systems should allow for variation in hand size and provide enough room for assistive devices to be used.

These seven principles give us a good starting point in considering universal design. They are not all equally applicable to all situations, of course. For example, principles six and seven would be vital in designing an information booth but less important in designing word-processing software. But they provide a useful checklist of considerations for designers, together with guidelines on how each principle can be achieved. It is interesting to note that these principles are closely related to the ones we met in Chapter 7, in the context of general user-centered design rules, indicating again that universal design is fundamentally good design for all.

10.3 MULTI-MODAL INTERACTION

As we have seen in the previous section, providing access to information through more than one mode of interaction is an important principle of universal design. Such design relies on *multi-modal interaction*.

As we saw in Chapter 1, there are five senses: sight, sound, touch, taste and smell.

Sight is the predominant sense for the majority of people, and most interactive systems consequently use the visual channel as their primary means of presentation, through graphics, text, video and animation.

However, sound is also an important channel, keeping us aware of our surroundings, monitoring people and events around us, reacting to sudden noises, providing clues and cues that switch our attention from one thing to another. It can also have an emotional effect on us, particularly in the case of music. Music is almost completely an auditory experience, yet is able to alter moods, conjure up visual images, evoke atmospheres or scenes in the mind of the listener.

Touch, too, provides important information: tactile feedback forms an intrinsic part of the operation of many common tools – cars, musical instruments, pens, anything that requires holding or moving. It can form a sensuous bond between individuals, communicating a wealth of non-verbal information.

Taste and smell are often less appreciated (until they are absent) but they also provide useful information in daily life: checking if food is bad, detecting early signs of fire, noticing that manure has been spread in a field, pleasure.

Examples of the use of sensory information are easy to come by (we looked at some in Chapter 1), but the important point is that our everyday interaction with each other and the world around us is multi-sensory, each sense providing different information that informs the whole. Since our interaction with the world is improved by multi-sensory input, it makes sense that interactive systems that utilize more than one sensory channel will also provide a richer interactive experience. In addition, such multi-sensory or multi-modal systems support the principle of redundancy required for universal design, enabling users to access the system using the mode of interaction that is most appropriate to their abilities.

The majority of interactive computer systems are predominantly visual in their interactive properties; often WIMP based, they usually make use of only rudimentary sounds while adding more and more visual information to the screen. As systems become more complex, the visual channel may be overloaded if too much information is presented all at once. This may lead to frustration or errors in use. By utilizing the other sensory channels, the visual channel can be relieved of the pressure of providing all the information required and so interaction should improve. The use of multiple sensory channels increases the *bandwidth* of the interaction between the human and the computer, and it also makes human–computer interaction more like the interaction between humans and their everyday environment, perhaps making the use of such systems more natural. However, it should always be remembered that multi-modal interaction is not just about enhancing the richness of the interaction, but also about redundancy. Redundant systems provide the *same* information through a range of channels, so, for example, information presented graphically is also captioned in readable text or speech, or a verbal narrative is provided with text captions. The aim is to provide at least an equivalent experience to all, regardless of their primary channel of interaction.

Usable sensory inputs

In computing, the visual channel is used as the predominant channel for communication, but if we are to use the other senses we have to consider their suitability and the nature of the information that they can convey.

The use of sound is an obvious area for further exploitation. There is little doubt that we use hearing a great deal in daily life, and so developing its application to the interface may be beneficial. Sound is already used, to a limited degree, in many interfaces: beeps are used as warnings and notification, recorded or synthesized speech and music are also used. Tactile feedback, as we have already seen, is also important in improving interactivity and so this represents another sense that we can utilize more effectively. However, taste and smell pose more serious problems for us. They are the least used of our senses, and are used more for receiving information than for communicating it. There are currently very few ways of implementing devices that can generate tastes and smells, and so these two areas are not supported. Whether this is a serious omission remains to be seen, but the tertiary nature of those senses tends to suggest that their incorporation, if it were possible, would focus on specialist applications, for example, in enhancing virtual reality systems.

Even if we do not use other senses in our systems, it is certainly worth thinking about the nature of these senses and what we gain from them as this will improve our understanding of the strengths and weaknesses of visual communication [96].

The next sections of this chapter will look at some of the alternative modes of human–computer communication, concentrating particularly on sound, touch, handwriting and gesture. We will consider how each mode can be used to create richer interaction and provide redundancy.

10.3.1 Sound in the interface

Sound is an important contributor to usability. There is experimental evidence to suggest that the addition of audio confirmation of modes, in the form of changes in keyclicks, reduces errors [237]. Video games offer further evidence, since experts tend to score less well when the sound is turned off than when it is on; they pick up vital clues and information from the sound while concentrating their visual attention on different things. The dual presentation of information through sound and vision supports universal design, by enabling access for users with visual and hearing impairments respectively. It also enables information to be accessed in poorly lit or noisy environments. Sound can convey transient information and does not take up screen space, making it potentially useful for mobile applications.

However, in spite of this, the auditory channel is comparatively little used in standard interfaces, and where it is used it is often peripheral to the interaction. Information provision is predominantly visual. There is a danger that this will overload the visual channel, demanding that the user attend to too many things at once and select appropriate information from a mass of detail in the display. Reliance on visual information forces attention to remain focussed on the screen, and the persistence of visual information means that even detail that is quickly out of date may remain on display after it is required, cluttering the screen further. It also presents significant problems for people with visual impairment, whose access to applications can be severely restricted by solely visual output. More widespread effective use of sound in the interface would alleviate these problems. There are two types of sound that we could use: speech and non-speech.

Speech in the interface

Language is rich and complex. We learn speech naturally as children 'by example' – by listening to and mimicking the speech of those around us. This process seems so effortless that we often do not appreciate its complex structures, and it is not until we attempt to learn a new language later in life, or to make explicit the rules of the one we speak, that the difficulties inherent in language understanding become apparent. This complexity makes speech recognition and synthesis by computer very difficult.

Structure of speech If we are fully to appreciate the problems involved with the computer-based recognition and generation of speech, we need first to understand the basic structure of speech. We will use English to illustrate but most other languages have similar issues.

The English language is made up of 40 *phonemes*, which are the atomic elements of speech. Each phoneme represents a distinct sound, there being 24 consonants and 16 vowel sounds. Language is more than simple sounds, however. Emphasis, stress, pauses and pitch can all be used to alter the meaning and nature of an utterance, a common example being the rise in pitch at the end of a sentence to indicate a question in English. This alteration in tone and quality of the phonemes is termed *prosody* and is used, in addition to the actual words, to convey a great deal of meaning and emotion within a sentence. Prosodic information gives language its richness and texture, but is very difficult to quantify. Owing to the manner in which sound is produced in the vocal tract, mouth and nose of the speaker, the limitation in response speed means that phonemes sound differently when preceded by different phonemes. This is termed *co-articulation*, and the resulting differences in sound can be used to construct a set of *allophones*, which represent all the different sounds within the language. Ignoring prosodic information, the concatenation of allophones together should produce intelligible, articulate speech. However, depending on the analysis of language used, and the regional accent studied, there are between 120 and 130 allophones. These, in turn, can be formed into *morphemes*, which represent the smallest unit of language that has meaning. They are the basic building blocks of language rather than of speech. Morphemes can be either parts of words or whole words, and they are built into sentences using the rules of grammar of the language.

Even being able to decompose sentences into their basic parts does not mean that we can then understand them: the syntax (structure) only serves as a standard foundation upon which the semantics (meaning) is based. We are rarely aware of the complex structure of speech, and concentrate on extracting the meaning from the sentences we hear, rather than decomposing the sounds into their constituent parts.

Speech recognition There have been many attempts at developing speech recognition systems, but, although commercial systems are now commonly and cheaply available, their success is still limited to single-user systems that require considerable training.

The complexity of language is one barrier to success, but there are other, more practical, problems also associated with the automatic recognition of the spoken word. Background noise can interfere with the input, masking or distorting the information, while speakers can introduce redundant or meaningless noises into the information stream by repeating themselves, pausing or using ‘continuation’ noises such as ‘ummm’ and ‘errr’ to fill in gaps in their usual speech. Variations between individuals also cause problems; people have unique voices, and systems that are successful are tuned to be sensitive to minute variations in tone and frequency of the speaker’s voice – new speakers present different inflections to the system, which then fails to perform as well. A more serious problem is caused by regional accents, which vary considerably. This strong variation upsets the trained response of the recognition system. More serious still is the problem posed by different languages: everything from phonemes up can be different.

The phonetic typewriter

One early successful speech-based system is the *phonetic typewriter*. This uses a neural network that clusters similar sounds together (see Figure 10.1).

Designed to produce typed output from speech input in Finnish, it is trained on one particular speaker, and then generalizes to others. However, its performance with speakers other than the one on which it was trained is noticeably poorer, and it relies on a large dictionary of minute variations to supplement its general transcription mechanism. Without the dictionary, it achieves a significantly lower recognition rate.

One reason that the phonetic typewriter was able to achieve acceptable levels of recognition and transcription is that Finnish is a phonetic language, that is one which is spelt as it sounds. There are other phonetic languages, for example Welsh, but most languages do not have such a straightforward mapping between sound and text. Think of English words such as 'wait' and 'weight' or 'one' and 'won', for example.

Puzzle: How do you pronounce 'ghuti'? (Answer on the web pages!)

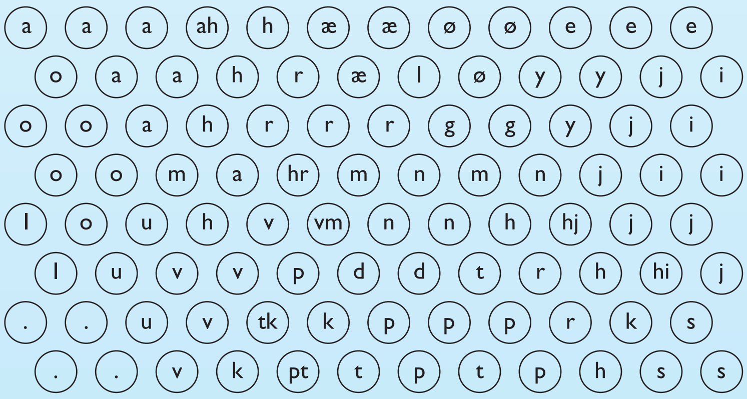


Figure 10.1 The phonetic typewriter

Speech recognition offers another mode of communication that may be used to supplement existing channels or be the primary one. When a user's hands are already occupied, such as in a factory, speech may prove to be the ideal input medium. Speech input does not require the use of a cumbersome keyboard and so in lightweight mobile situations there is a potential role for such systems. It also provides an alternative means of input for users with visual, physical and cognitive impairment as we will see later. Single-user, limited vocabulary systems can work satisfactorily, but the current success rate of recognition for general users and unconstrained language is still low.

Despite its limitations there are commercial systems employing speech recognition. Speech-based word processors are easily available and several computers use speech input as a marketing feature. Telephone-based systems also use speech, but they face a more difficult task as they must be speaker independent. At the simplest end, some systems ask you to speak an extension number, but, as tone dialing becomes universal, the advantage of this over typing the number is dubious! Other systems make more active use of voice, including information systems for airline bookings. These more sophisticated systems work because they are *interactive*: the system reflects back to the user its interpretation of the speech input, allowing the user to enter into a dialog to correct any errors. This is precisely what happens in normal conversation – we don't get it right all the time.

Speech synthesis Complementary to speech recognition is speech synthesis. The notion of being able to converse naturally with a computer is an appealing one for many users, especially those who do not regard themselves as computer literate, since it reflects their natural, daily medium of expression and communication. However, there are as many problems in speech synthesis as there are in recognition. The most difficult problem is that we are highly sensitive to variations and intonation in speech, and are therefore intolerant of imperfections in synthesized speech. We are so used to hearing natural speech that we find it difficult to adjust to the monotonic, non-prosodic tones that synthesized speech can produce. In fact, most speech synthesizers can deliver a degree of prosody, but in order to decide what intonation to give to a word, the system must have an understanding of the domain. So an effective automatic reader would also need to be able to understand natural language, which is difficult. However, for 'canned' messages and responses, the prosody can be hand coded yielding much more acceptable speech.

Synthesized speech also brings other problems. Being transient, spoken output cannot be reviewed or browsed easily. It is intrusive, requiring either an increase in noise in the office environment or the use of headphones, either of which may be too large a price to pay for the benefits the system may offer.

However, there are some application areas in which speech synthesis has been successful. For users who are blind or partially sighted, synthesized speech offers an output medium which they can access. Screen readers are software packages that read the contents of a computer screen, using synthesized speech. Modern screen readers read more than simply the text on the screen. They read exactly what they find including icons, menus, punctuation and controls. They also read events, such as dialog boxes opening, so that they can be used with graphical interfaces.

Speech synthesis is also useful as a communication tool to assist people with physical disabilities that affect their speech. Here speech synthesis needs to produce output that is as natural as possible with as little input effort from the user as possible, perhaps using a simple switch. Human conversation is rapid and complex, making this a significant challenge. Most communication tools of this type use predefined messages, enabling the user to select a message appropriate to the context quickly and easily.

Design Focus

Designing websites for screen readers

While screen readers provide users with visual impairments access to standard interfaces and software, web access can be more problematic. Screen readers can only read textual elements of web pages, so graphics and scripts cannot be interpreted. It is therefore important that web designers take account of these limitations and design sites to be accessible to people using screen readers. For example, HTML 'alt' tags should always be used for necessary graphics, and text alternative menus and navigation controls provided.

In addition, most read text across the page, so text arranged in columns can become garbled. For example, consider text arranged in two columns, such as the opening lines of these nursery rhymes:

```
Jack and Jill went up the hill      Mary had a little lamb
To fetch a pail of water            Its fleece was white as snow
```

With some screen readers, this text would be read as 'Jack and Jill went up the hill Mary had a little lamb To fetch a pail of water Its fleece was white as snow' – clearly nonsense. Where possible, text should make sense when read across a page.

Users of screen readers may also find it difficult to follow links embedded in text, particularly where there are several in a block of text. It can therefore be helpful to provide links to the main sections in a clear location where they will be read horizontally, such as at the top of the page.

If you want to experience something of what it is like to access the web using a screen reader, try the simulation produced by the Web Accessibility in Mind project, available at www.webaim.org/simulations/screenreader (accessed March 2003). Their site also contains a wealth of information about web accessibility.

Another useful resource to help you design accessible websites is the Web Accessibility Initiative's (WAI) checklist for accessibility. This is available on their website www.w3.org (accessed March 2003).

A tool that you can use to test your websites for accessibility is Bobby from Watchfire (bobby.cast.org, accessed March 2003). Bobby is a web-based accessibility evaluation tool that highlights problem areas of a given website and encourages accessible solutions. It provides suggestions for improvement using the guidelines provided by the WAI and the US Government's Section 508 legislation. Using Bobby, web designers can test their sites for accessibility and get advice for resolving any problems before the site goes live.

Used as a supplement to other output channels, speech can also enhance applications where the user's visual attention is focussed elsewhere, such as warnings in aircraft cockpits and, more recently, in cars. We will return to some of these applications later in the chapter.

Uninterpreted speech Speech does not have to be recognized by a computer to be useful in the interface. Fixed pre-recorded messages can be used to supplement or replace visual information. Recordings have natural human prosody and pronunciation, although quality is sometimes low. Segments of speech can be used together to construct messages, for example the announcements in many airports and railway stations.

Recordings of users' speech can also be very useful, especially in collaborative applications, for example many readers will have used voicemail systems. Also, recordings can be attached to other artifacts as *audio annotations* in order to communicate with others or to remind oneself at a later time. For example, audio annotations can be attached to Microsoft Word documents.

When recordings are replayed, they can be digitally speeded up. If you simply play an audio recording faster, the pitch rises – and human speech ends up sounding rather like Mickey Mouse. However, *digital signal-processing* techniques can accelerate a recording while keeping the same pitch. Speech can be played back at up to twice the normal rate without any loss of comprehensibility. This can be used in a telephone help desk where a pre-recorded message asks the enquirer to state his problem. The problem can then be replayed at an accelerated rate to the operator, reducing the operator time per enquiry. The utility of such methods needs careful analysis, however. The operator may often begin to act on a message while it is still playing, hence reducing any gain from faster playback. Furthermore, reduced interactivity may lead to more misunderstandings, and the enquirer's waiting time may be increased.

Design Focus

Choosing the right kind of speech

If you include speech input in an interface you must decide what level of speech interaction you wish to support:

recording simply recording and replaying messages or annotations;

transcription turning speech into text as in a word processor;

control telling the computer what to do: for example, 'print this file'.

Each level has its own problems and advantages; for example, control only requires a limited vocabulary, but is more dangerous: 'I said print not delete . . . !' However, the biggest problem arises if you try to mix these levels. In text we use quotes to make such distinctions, but they are hard in speech: 'insert the word "delete" before the word "before"'.

In fact, for general interface use, speech is best mixed with other modes of communication as happens in everyday life. For example, in a word processor you may use a tablet and pen to ring a word and then say 'move this word to here' as you tap the pen at the target location. This is exactly what you would do when talking through corrections to a document with someone.

Non-speech sound

We have considered the use of speech in the interface, but non-speech sounds can offer a number of advantages. As speech is serial, we have to listen to most of a sentence before we understand what is being said. Non-speech sounds can often be assimilated much more quickly. Speech is language dependent – a speech-based system requires translation for it to be used for another language group. The meaning of non-speech sounds can be learned regardless of language. Speech requires the user's attention. Non-speech sound can make use of the phenomenon of auditory adaptation: background sounds are ignored unless they change or cease. However, a disadvantage is that non-speech sounds have to be learned, whereas the meaning of a spoken message is obvious (at least to a user who is familiar with the language used).

Non-speech sound can be used in a number of ways in interactive systems. It is often used to provide transitory information, such as indications of network or system changes, or of errors. It can also be used to provide status information on background processes, since we are able to ignore continuous sounds but still respond to changes in those sounds. Users of early home computers with their noisy power supplies, and computer operators listening to the chatter of the printer and the spinning of disks and tape drives, both report that they are able to tell what stage a process is at by the characteristic sounds that are made.

Non-speech sound can also be used to provide a second representation of actions and objects in the interface to support the visual mode and provide confirmation for the user. It can be used for navigation round a system, either giving redundant supporting information to the sighted user or providing the primary source of information for the visually impaired. Experiments on auditory navigation [290] have demonstrated that auditory clues are adequate for a user to locate up to eight targets on a screen with reasonable speed and accuracy. This suggests that there is little reason for ignoring the role of sound in interfaces on the grounds that it may be too vague or inaccurate.

But what kind of non-speech sounds should we use in the interface? There are two alternatives: using sounds that occur naturally in the world and using more abstract generated sounds. We will consider an example of each type.

Auditory icons *Auditory icons* [141] use natural sounds to represent different types of objects and actions in the interface. The SonicFinder [142] for the Macintosh was developed from these ideas, to enhance the interface through redundancy. Natural sounds are used because people recognize the source of a sound and its behavior rather than timbre and pitch [364]. For example, a noise will be identified as glass breaking or a hollow pipe being tapped. Such recognition is quite sophisticated: we can identify not only the source of a sound (e.g. tapping a pipe) but characteristics of the sound source (e.g. whether the pipe is hollow or solid).

In the SonicFinder, auditory icons are used to represent desktop objects and actions. So, for example, a folder is represented by a papery noise, and throwing something in the wastebasket by the sound of smashing. This helps the user to learn the sounds since they suggest familiar actions from everyday life. However, this advantage also creates a problem for auditory icons. Some objects and actions do not have obvious, naturally occurring sounds that identify them. In these cases a sound effect can be created to suggest the action or object but this moves away from the ideal of using familiar everyday sounds that require little learning. For example, copying has no immediate analog sound and in the SonicFinder it is indicated by the sound of pouring a liquid into a receptacle, with the pitch rising to indicate the progress of the copying.

SharedARK and ARKola

Natural sounds have been used to model environments such as a physics laboratory [145], called *SharedARK* (Shared Alternate Reality Kit) and a virtual manufacturing plant, *ARKola* [147]. In SharedARK, multiple users could perform physics experiments in a virtual laboratory. Sound was used in three different ways: as confirmation of actions, for status information and as aids to navigation. Confirmatory sounds use similar principles to the SonicFinder, providing redundant information that increases feedback. Process and state information sounds exist on two levels, global and local. Global sounds represent the state of the whole system and can be heard anywhere, while local sounds are specific to particular experiments and alter when the user changes from one experiment to another. Navigational information is provided by soundholders, which are auditory landmarks. They can be placed anywhere in the system and get louder as the user moves towards them, decreasing in volume when moving away. This allows the user to wander through an arena much greater than the size of the screen without getting lost and lets him return to specific areas very easily by returning to the soundholder.

In ARKola, a soft drinks factory was modeled, with two users attempting to optimize the factory's output, working remotely from each other and using an audio/video link. Input machines supplied raw materials while output machines capped the bottles and shipped them out. Each machine had an on/off switch and a rate control, with a sound that indicated its status; for example, the bottle dispenser made the sound of clinking glass, with a rhythm that indicated its operating speed. Splashing sounds indicated spilled liquids, while breaking glass showed that bottles were being lost. The users monitored the status of the plant by listening to the auditory clues, and were able to help each other more effectively, since they found it easier to monitor their own machines without having to spend time looking at them, and could hear when something had gone wrong with their partner's part of the system.

Non-speech sounds such as this can convey a lot of meaning very economically. A file arrives in a mailbox and, being a large file, it makes a weighty sound. If it is a text file it makes a rustling noise, whereas a compiled program may make a metallic clang. The sound can be muffled or clear, indicating whether the mailbox is hidden by other windows or not, while the direction of the sound would indicate the position of the mailbox icon on the screen. If the sound then echoes, as it would in a large, empty room, the system load is low. All this information can be presented in a second or so.

Worked exercise *Think of a set of naturally occurring sounds to represent the operations in a standard drawing package (for example, draw, move, copy, delete, rotate).*

Answer This can exercise the imagination! Are there natural analogies? For example, does the physical action, say, of drawing have a sound associated with it? The sound of a pencil on paper may be appropriate but is it identifiable? Similarly, a photocopier whirring could represent the copy operation, and tearing paper delete. Rotate and move are more difficult since the physical operation is not associated with a sound. Perhaps direction and movement can be indicated by sounds becoming nearer or more distant?

Earcons

An alternative to using natural sounds is to devise synthetic sounds. *Earcons* [36] use structured combinations of notes, called *motives*, to represent actions and objects (see Figure 10.2). These vary according to rhythm, pitch, timbre, scale and volume. There are two types of combination of earcon. *Compound earcons* combine different motives to build up a specific action, for example combining the motives for 'create' and 'file'. *Family earcons* represent compound earcons of similar types. As an example, operating system errors and syntax errors would be in the 'error' family. In this way, earcons can be hierarchically structured to represent menus. Earcons are easily grouped and refined owing to their compositional and hierarchical nature, but they require learning to associate with a specific task in the interface since there is an

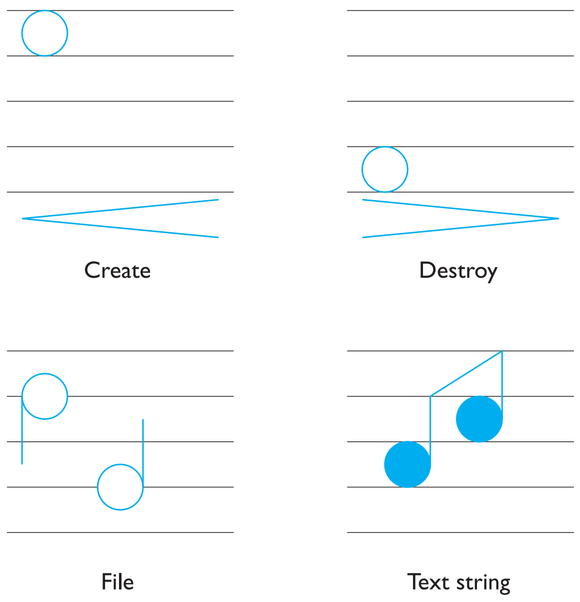


Figure 10.2 Earcons (after Blattner [36], reprinted by permission of Lawrence Erlbaum Associates, Inc.)

arbitrary mapping. Conversely, auditory icons have a semantic relationship with the function that they represent, but can suffer from there being no appropriate sound for some actions.

Earcons provide a structured approach to designing sound for the interface, but can users learn the sounds adequately, and what factors influence their use? Evidence suggests that people can learn to recognize earcons, and that the most important element in distinguishing different sounds is timbre, the characteristic quality of the sound produced by different instruments and voices [47]. Other factors such as pitch, rhythm and register should be used to supplement timbre in creating distinctive sets of musical earcons. Interestingly, the user's musical ability appears to have little effect on his ability to remember earcons: users were able to identify around 80% of earcons from hierarchically ordered sets of 30 or more, regardless of their musical background [45]. It is also possible to create compound earcons by playing sounds in parallel as well as serially. This obviously reduces the time taken to hear the sound but does not affect the user's accuracy [45].

10.3.2 Touch in the interface

We have already considered the importance of touch in our interaction with our environment, in Chapter 1. Touch is the only sense that can be used to both send and receive information. Although it is not yet widely used in interacting with computers, there is a significant research effort in this area and commercial applications are becoming available.

The use of touch in the interface is known as *haptic interaction*. Haptics is a generic term relating to touch, but it can be roughly divided into two areas: cutaneous perception, which is concerned with tactile sensations through the skin; and kinesthetics, which is the perception of movement and position. Both are useful in interaction but they require different technologies.

In Chapter 2, Section 2.6.3, we considered a number of examples of haptic devices, including some based on vibration against the skin (cutaneous) and others on resistance or force feedback (kinesthethic). They facilitate perception of properties such as shape, texture, resistance and temperature as well as comparative spatial properties such as size, height and position. This means haptics can provide information on the character of objects in the interface, as well as more realistic simulations of physical activities, either for entertainment or for training.

In this section, we will look in a little more detail at some of the different types of haptic devices and consider, in particular, the role of haptics in universal design. As we will see, touch can provide both a primary source of information for users with visual impairments and a richer multi-modal experience for sighted users.

One example of a tactile device is an electronic – or soft – braille display. Braille displays are made up of a number of cells (typically between 20 and 80), each containing six or eight electronically controlled pins that move up and down to produce braille representations of characters displayed on the screen. Whereas printed braille normally has six dots per cell, electronic braille typically has eight pins, with the extra two representing additional information about that cell, such as cursor position and character case.

Electronic braille displays benefit from two factors: a well-established tactile notation (braille) and a user group with expertise in using this notation. But can similar techniques be used to provide more generic tactile feedback, such as to display graphics? The problem with using raised pins for this type of display is the resolution required. Braille requires only six or eight pins; a graphical display would require many more, which raises the problem of fitting the necessary number of fast actuators (to move the pins) into a few cubic centimeters. This presents a serious engineering challenge.

The other main type of haptic device is the force feedback device, which provides kinesthetic information back to the user, allowing him to feel resistance, textures, friction and so on. One of the most commonly used examples is the PHANTOM range, from SensAble Technologies. The PHANTOM provides three-dimensional

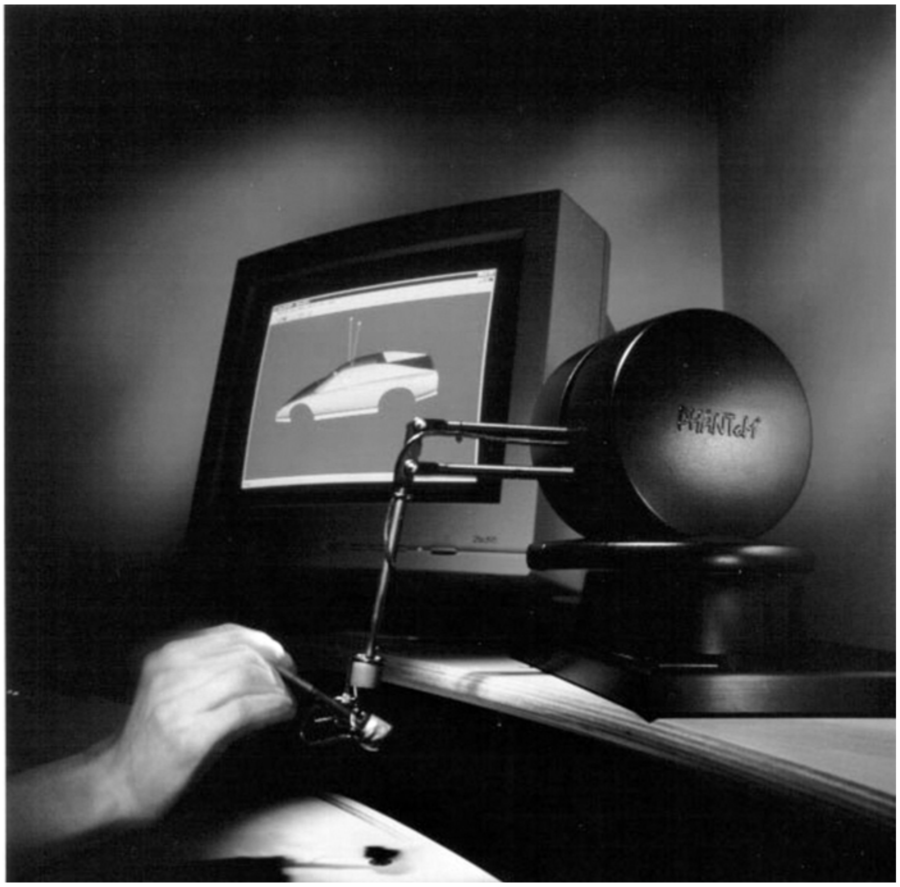


Figure 10.3 A PHANTOM Premium 1.5 haptic device. Source: Courtesy of SensAble Technologies

force feedback, allowing users to touch virtual objects. As well as offering the functionality of the mouse, in addition, the user's movement is monitored by optical sensors on the device, and these, together with models of the virtual objects, are used to calculate the forces applied back to the user. The user therefore can feel the outline and resistance of objects, their texture and position. This type of device has potential application for simulations and training situations where touch is important, such as medicine. It can also be used to provide a haptic 'image' of an interface, providing the user with information about the objects and their functionality based on how they feel. This offers another channel of information, which enhances the richness of the interaction and makes the design more universal.

At present, the hardware needed to support haptic interaction is prohibitively expensive for most users. But this is liable to change as the applications become more widespread and commercially viable.

10.3.3 Handwriting recognition

Like speech, we consider handwriting to be a very natural form of communication. The idea of being able to interpret handwritten input is very appealing, and handwriting appears to offer both textual and graphical input using the same tools. There are problems associated with the use of handwriting as an input medium, however, and in this section we shall consider these. We will first look at the mechanisms for capturing handwritten information, and then look at the problems of interpreting it.

Technology for handwriting recognition

The major piece of technology used to capture handwriting is the digitizing tablet, explained in more detail in Chapter 2. Free-flowing strokes made with a pen are transformed into a series of coordinates, approximately one every 1/50th of a second (or at the sampling rate of the digitizer). Rapid movements produce widely spaced dots, in comparison with slow movements: this introduces immediate errors into the information, since the detail of the stroke between dots is lost, as is the pressure information.

Design Focus

Apple Newton

The Apple Newton was the first popular pen-based computer. Other systems, such as the GO PenPoint computer, were available earlier, but did not achieve a significant breakthrough. One reason is that the Newton targeted the organizer market where miniature keyboards were difficult to use (and anyway managers don't use them!). Also this niche market did not demand large amounts of text input, and the graphical interface made it easy to do non-text-based tasks.

Handwriting recognition was acceptable for a number of reasons: the base algorithm achieved a reasonable level of writer-independent recognition; the algorithm was adaptive – it learned the characteristics of the owner during use; and it was word based, so that idiosyncrasies in connected writing could be learnt for common words. But, most important, it was *interactive*. After a word was written, the Newton printed its interpretation of the word; if it was wrong you could try again or correct it letter by letter. This gave the system a chance to learn and meant that errors were not fatal!

In fact, although it has survived, the Apple Newton, like many devices employing novel input techniques, did not achieve the level of success one might have envisaged. This may be because it arrived at the same time as portable computers became *really* portable, and perhaps because the Apple Newton was only suitable for large pockets (of both a sartorial and financial nature). Smaller organizers with both pen-based input and small keyboards are now available, and it remains to be seen whether these achieve the market breakthrough this technology promises.

Digitizing tablets have been refined by incorporating a thin screen on top to display the information, producing *electronic paper*. Advances in screen technology mean that such devices are small and portable enough to be realistically useful in handheld organizers such as the Apple Newton. Information written onto the digitizer can simply be redisplayed, or stored and redisplayed for further reference. However, while this has limited use in itself, systems are most useful when they are able to interpret the strokes received and produce text. It is this recognition that we will look at next.

Recognizing handwriting

The variation between the handwriting of individuals is large (see Figure 10.4); moreover, the handwriting of a single person varies from day to day, and evolves over the years.

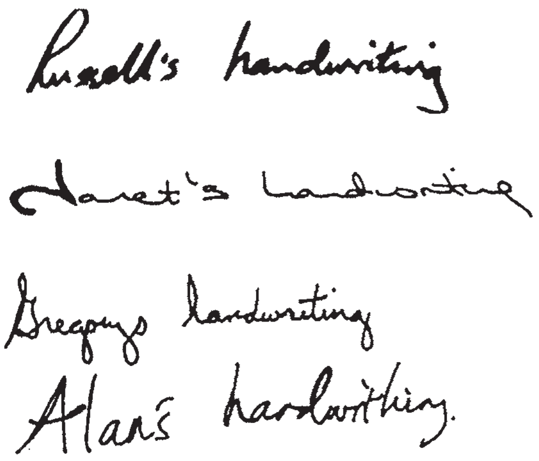


Figure 10.4 Handwriting varies considerably

These problems are reminiscent of those already discussed in speech recognition, and indeed the recognition problem is not dissimilar. The equivalent of co-articulation is also prevalent in handwriting, since different letters are written differently according to the preceding and successive ones. This causes problems for recognition systems, which work by trying to identify the lines that contain text, and then to segment the digitized image into separate characters. This is so difficult to achieve reliably that there are no systems in use today that are good at general cursive script recognition. However, when letters are individually written, with a small separation, the success of systems becomes more respectable, although they have to be trained to recognize the characteristics of the different users. If tested on an untrained person, success is limited again. Many of the solutions that are being attempted in speech recognition are also being tried in handwriting recognition systems, such as whole-word recognition, the use of context to disambiguate characters, and neural networks, which learn by example.

10.3.4 Gesture recognition

Gesture is a component of human–computer interaction that has become the subject of attention in multi-modal systems. Being able to control the computer with certain movements of the hand would be advantageous in many situations where there is no possibility of typing, or when other senses are fully occupied. It could also support communication for people who have hearing loss, if signing could be 'translated' into speech or vice versa. But, like speech, gesture is user dependent, subject to variation and co-articulation. The technology for capturing gestures is expensive, using either computer vision or a special dataglove (see Chapter 2). The dataglove provides easier access to highly accurate information, but is a relatively intrusive technology, requiring the user to wear the special Lycra glove. The interpretation of the sampled data is very difficult, since segmenting the gestures causes problems. A team from Toronto [131] has produced a gesture recognition system that translates hand movements into synthesized speech, using five neural networks working in parallel to learn and then interpret different parts of the inputs.

The Media Room at MIT uses a different approach in order to incorporate gestures into the interaction. The Media Room has one wall that acts as a large screen, with smaller touchscreens on either side of the user, who sits in a central chair. The user can navigate through information using the touchscreens, or by joystick, or by voice. Gestures are incorporated by using a position-sensing cube attached to a wristband worn by the user. The *put that there* system uses this gestural information coupled with speech recognition to allow the user to indicate what should be moved where by pointing at it. This is a much more natural form of interaction than having to specify verbally what it is that has to be moved and describing where it has to go, as well has having the advantage of conciseness. Such a short, simple verbal statement is much more easily interpreted by the speech recognition system than a long and complex one, with the resolution of ambiguity done by interpreting the other mode of interaction, the gesture. Each modality supports the other.

10.4 DESIGNING FOR DIVERSITY

We noted in Chapter 1 that, although we can make general observations about human capabilities, users in fact have different needs and limitations. Interfaces are usually designed to cater for the 'average' user, but unfortunately this may exclude people who are not 'average'. As we saw in the introduction to this chapter, people are diverse and there are many factors that must be taken into account if we are to come close to universal design.

In this section, we will consider briefly some of these factors and the particular challenges that each raises. We will consider three key areas: disability, age and culture.

10.4.1 Designing for users with disabilities

It is estimated that at least 10% of the population of every country has a disability that will affect interaction with computers. Employers and manufacturers of computing equipment have not only a moral responsibility to provide accessible products, but often also a legal responsibility. In many countries, legislation now demands that the workplace must be designed to be accessible or at least adaptable to all – the design of software and hardware should not unnecessarily restrict the job prospects of people with disabilities.

We will look briefly at sensory, physical and cognitive impairments and the issues they raise for interface design.

Visual impairment

The sensory impairment that has attracted the most attention from researchers, perhaps because it is potentially also one of the most debilitating as far as interaction is concerned, is visual impairment. The rise in the use of graphical interfaces reduces the possibilities for visually impaired users. In text-based interaction, screen readers using synthesized speech or braille output devices provided complete access to computers: input relied on touch-typing, with these mechanisms providing the output. However, today the standard interface is graphical. Screen readers and braille output are far more restricted in interpreting the graphical interface, as we saw in Section 10.3.1, meaning that access to computers, and therefore work involving computers, has been reduced rather than expanded for visually impaired people.

There are two key approaches to extending access: the use of sound and the use of touch. We have already considered these in Section 10.3 so we will summarize only briefly here.

A number of systems use sound to provide access to graphical interfaces for people with visual impairment. In Section 10.3.1 we looked at a range of approaches to the use of sound such as speech, earcons and auditory icons. All of these have been used in interfaces for blind users.

Soundtrack

Soundtrack is an early example of a word processor with an auditory interface, designed for users who are blind or partially sighted [118]. The visual items in the display have been given auditory analogs, made up of tones, with synthesized speech also being used. A two-row grid of four columns is Soundtrack's main screen (see Figure 10.5); each cell makes a different tone when the cursor is in it, and by using these tones the user can navigate around the system. The tones increase in pitch from left to right, while the two rows have different timbres. Clicking on a cell makes it speak its name, giving precise information that can reorient a user who is lost or confused. Double clicking on a cell reveals a submenu of items associated with the main screen item. Items in the submenu also have tones; moving down the menu causes the tone to fall whilst moving up makes it rise. A single click causes the cell to speak its name, as before, whilst double clicking executes the associated action.

Soundtrack allows text entry by speaking the words or characters as they are entered, with the user having control over the degree of feedback provided. It was found that users tended to count the different tones in order to locate their position on the screen, rather than just listen to the tones themselves, although one user with musical training did use the pitch.

Soundtrack provides an auditory solution to representing a visually based word processor, though the results are not extensible to visual interfaces in general. However, it does show that the human auditory system is capable of coping with the demands of highly interactive systems, and that the notion of auditory interfaces is a reasonable one.

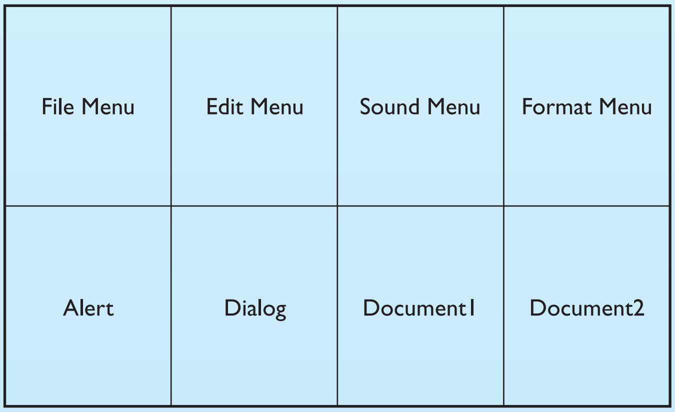


Figure 10.5 The screen division in Soundtrack. Source: Courtesy of Alistair D. N. Edwards

Soundtrack (see the box above) was an early example of the use of non-speech sound to provide an auditory interface to a word processor. A major limitation of this application was the fact that it was a specialized system; it could not be used to augment commercially available software. Ideally, users with disabilities should have

Design Focus

Mathematics for the blind

Solve the following equation: $3(x - 2) + 4 = 7 - 2(3 - x)$.

Did you do it in your head or use a piece of paper? When an equation is even slightly complex the instant response of a sighted person is to reach for paper and pencil. The paper acts as an *external memory*, allowing you to record and recall previous steps in a calculation. Blind children learning mathematics have to perform nearly all such calculations in their head, putting them at a severe disadvantage.

Mathtalk is a system developed as part of a European project to create a mathematics workstation for blind people [330]. It uses speech synthesis to speak formulae, and keyboard input to navigate and manipulate them. The first stage, simply speaking a formula out loud, is complex in itself. Given the spoken equation 'three x plus four equals seven', how do you know whether this is '$3x + 4 = 7$' or '$3(x + 4) = 7$'? To make it unambiguous one could say the latter as 'three open bracket x plus four close bracket equals seven', but this soon becomes very tedious. In fact, when reading mathematics people use several cues in their speech: longer and shorter gaps between terms, and prosody: rising and falling pitch (see Figure 10.6). The Mathtalk system includes a set of rules for generating such patterns suitable for most equations.

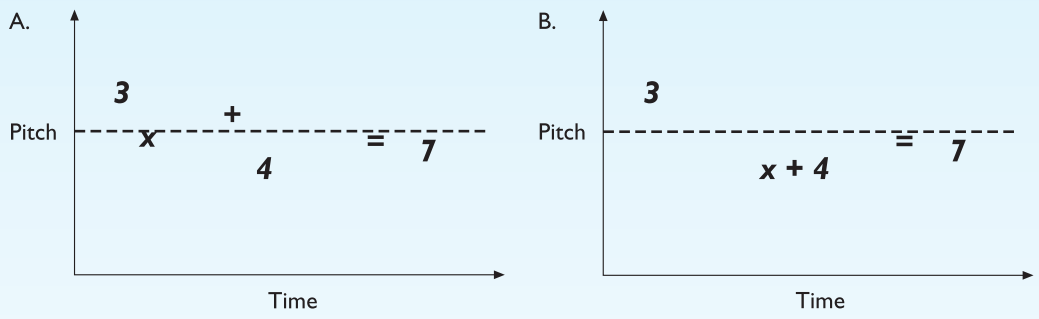


Figure 10.6 Pausing and pitch help distinguish between two expressions

Visual interaction with paper isn't just at the level of reading and writing whole equations. Recall from Chapter 1 that reading usually includes *regressions* where our eyes move backwards as well as forwards through text. Also, when seeing graphical material (remember that mathematics makes heavy use of brackets, symbols, superscripts, etc.), we rely on getting a quick feel for the material at a glance before examining it in detail. Both of these factors are crucial when reading an equation and so Mathtalk supports rapid keyboard-based navigation *within* each equation, and *algebra earcons*, short motives based on the rise and fall of the prosody of an equation.

Notice that Mathtalk uses *keyboard* input combined with speech output. Speech input is slow and error-prone compared with a keyboard. Braille output can also be used for mathematics, but only a small percentage of blind people read braille. Choosing the right input and output devices requires a deep knowledge of the user population and careful analysis of the intended tasks.

access to the same applications as anyone else. Outspoken is a Macintosh application that uses synthetic speech to make other Macintosh applications available to visually impaired users. A common problem with this and other screen readers and talking browsers (see Section 10.3.1) is the sheer amount of information represented. Browsing is difficult and all of the information must be held in the head of the user, putting a heavy load on memory.

A more recent development is the use of touch in the interface. As we saw in Section 10.3.2, there are two key approaches to this, both of which can be used to support people with visual impairment. Tactile interaction is already widely used in electronic braille displays, which represent what is on the screen through a dynamic braille output. It could also be used to provide more information about graphics and shape, if the engineering challenges of building higher resolution tactile devices can be overcome. Force feedback devices also have the potential to improve accessibility to users with visual impairment, since elements in the interface can be touched, and edges, textures and behavior used to indicate objects and actions. A limitation of this technology at present is that objects must be rendered using specialist software in order for the devices to calculate the appropriate force to apply back to the user. This again represents a move away from use of generic applications to specialist applications. However, it is likely that major applications will become ‘haptic enabled’ in the near future.

Hearing impairment

Compared with a visual disability where the impact on interacting with a graphical interface is immediately obvious, a hearing impairment may appear to have little impact on the use of an interface. After all, it is the visual not the auditory channel that is predominantly used. To an extent this is true, and computer technology can actually enhance communication opportunities for people with hearing loss. Email and instant messaging are great levellers and can be used equally by hearing and deaf users alike.

Gesture recognition has also been proposed to enable translation of signing to speech or text, again to improve communication particularly with non-signers.

However, the increase in multimedia and the use of sound in interfaces has, ironically, created some access difficulties for people with hearing problems. Many multimedia presentations contain auditory narrative. If this is not supplemented by textual captions, this information is lost to deaf users. Captioning audio content, where there is not already a graphical or textual version, also has the advantage of making audio files easier and more efficient to index and search, which in turn enhances the experience of all users – a sure sign of good universal design!

Physical impairment

Users with physical disabilities vary in the amount of control and movement that they have over their hands, but many find the precision required in mouse control difficult. Speech input and output is an option for those without speech difficulties.

An alternative is the eyegaze system (Chapter 2), which tracks eye movements to control the cursor, or a keyboard driver that can be attached to the user's head. If the user is unable to control head movement, gesture and movement tracking can be used to allow the user control. If the user has limited use of a keyboard, a predictive system, such as the Reactive keyboard [157], can help, by anticipating the commands that are being typed and offering them for execution. This can cut the typing requirement considerably. Predictions are based on what the user has typed in the current session or a previous one. The predictions therefore anticipate within the context in which the user is currently working (for example, operating system commands, programming text or free text). Figure 10.7 shows an interaction using the Reactive keyboard.

```
$  mail                                              ↑N
   cd news                                           ↑W
   cd news                                           ↑N
   cd rk/papers/ieee.computer                        ↑L
   cd rk/papers/ieee.computer

$  emacs paper.tex                                   ↑L
   emacs paper.tex

$  rm paper.tex.CKP paper.tex.BAK                    ↑L
   rm paper.tex.CKP paper.tex.BAK

$  wc -w paper.tex                                   ↑L
   wc -w paper.tex

$  readnews -n comp.sources.unix                     ↑N
   mail                                              ↑W
   mail                                              ↑N
   mail bdarragh%uncamult.bitnet@ucnet.ucalgary.c    ↑L
   mail bdarragh%uncamult.bitnet@ucnet.ucalgary.c
```

User's dialog with the Reactive keyboard.
Only the last line in each group is actually executed.

Key	*Description*
↑C (control-C)	Accept the next predicted character
↑W	Accept the next predicted word
↑L	Accept the whole predicted line
↑N	Show the next alternative prediction
↑P	Show the previous alternative prediction

Reactive keyboard commands

Figure 10.7 An interaction using the Reactive keyboard. Source: Courtesy of Saul Greenberg

Speech impairment

For users with speech and hearing impairments, multimedia systems provide a number of tools for communication, including synthetic speech (see Section 10.3.1) and text-based communication and conferencing systems (see Chapter 19). Textual communication is slow, which can lower the effectiveness of the communication. Predictive algorithms have been used to anticipate the words used and fill them in, to reduce the amount of typing required. Conventions can help to provide context, which is lost from face-to-face communication, for example the 'smilie' :-), to indicate a joke. Facilities to allow turn-taking protocols to be established also help natural communication [256]. Speech synthesis also needs to be rapid to reflect natural conversational pace, so responses can be pre-programmed and selected using a single switch.

Dyslexia

Users with cognitive disabilities such as dyslexia can find textual information difficult. In severe cases, speech input and output can alleviate the need to read and write and allow more accurate input and output. In cases where the problem is less severe, spelling correction facilities can help users. However, these need to be designed carefully: often conventional spelling correction programs are useless for dyslexic users since the programs do not recognize their idiosyncratic word construction methods. As well as simple transpositions of characters, dyslexic users may spell phonetically, and correction programs must be able to deal with these errors.

Consistent navigation structure and clear signposting cues are also important to people with dyslexia. Color coding information can help in some cases and provision of graphical information to support textual can make the meaning of text easier to grasp.

Autism

Autism affects a person's ability to communicate and interact with people around them and to make sense of their environment. This manifests itself in a range of ways but is characterized by the *triad of impairments*:

1. Social interaction – problems in relating to others in a meaningful way or responding appropriately to social situations.
2. Communication – problems in understanding verbal and textual language including the use of gestures and expressions.
3. Imagination – problems with rigidity of thought processes, which may lead to repetitive behavior and inflexibility.

How might universal design of technology assist people with autism? There are two main areas of interest: communication and education.

Communication and social interaction are major areas of difficulty for people with autism. Computers, on the other hand, are often motivating, perhaps because they are relatively consistent, predictable and impersonal in their responses. The user is in control. Computer-mediated communication and virtual environments have been suggested as possible ways of enabling people with autism to communicate more easily with others, by giving the user control over the situation. Some people with autism have difficulties with language and may be helped by graphical representations of information and graphical input to produce text and speech. Again this is supported by providing redundancy in the design.

Computers may also have a role to play in education of children with autism, particularly by enabling them to experience (through virtual environments and games) social situations and learn appropriate responses. This can again provide a secure and consistent environment where the child is in control of his own learning. The use of computers to support people with autism in this way is still a new research area and it is likely that new software and tools will develop in the next few years.

10.4.2 Designing for different age groups

We have considered how people differ along a range of sensory, physical and cognitive abilities. However, there are other areas of diversity that impact upon the way we design interfaces. One of these is age. In particular, older people and children have specific needs when it comes to interactive technology.

Older people

The proportion of older people in the population is growing steadily. Contrary to popular stereotyping, there is no evidence that older people are averse to using new technologies, so this group represents a major and growing market for interactive applications. People are living longer, have more leisure time and disposable income, and older people have increased independence. These factors have all led to an increase in older users.

But the requirements of the older population may differ significantly from other population groups, and will vary considerably within the population group. The proportion of disabilities increases with age: more than half of people over 65 have some kind of disability. Just as in younger people with disabilities, technology can provide support for failing vision, hearing, speech and mobility. New communication tools, such as email and instant messaging, can provide social interaction in cases where lack of mobility or speech difficulties reduce face-to-face possibilities. Mobile technologies can be used to provide memory aids where there is age-related memory loss.

Some older users, while not averse to using technology, may lack familiarity with it and fear learning. They may find the terminology used in manuals and training books difficult to follow and alien (words like ‘monitor’ and ‘boot’, for example, may have a completely different meaning to an older person than a young person). Interests and concerns may also be different from younger users.

Once again, basic universal design principles are important here. Access to information must make use of redundancy and support the use of access technologies. Designs must be clear and simple and forgiving of errors. In addition, thought needs to be given to sympathetic and relevant training aimed at the user's current knowledge and skills.

In spite of the potential benefits of interactive technology to older people, very little attention has been paid to this area until recently. Researchers are now beginning to address issues such as how technology can best support older people, what the key design issues are, and how older people can be effectively included in the design process [46], and this area is likely to grow in importance in the future.

Children

Like older people, children have distinct needs when it comes to technology, and again, as a population, they are diverse. The requirements of a three year old will be quite different from those of a 12 year old, as will be the methods that can be used to uncover them. Children are, however, different from adults, and have their own goals and likes and dislikes. It is therefore important to involve them in the design of interactive systems that are for their use, though this in itself can be challenging as they may not share the designer's vocabulary or be able to verbalize what they think. Design approaches have therefore been developed specifically to include children actively as members of the design team. Alison Druin's Cooperative Inquiry approach [110] is based on contextual inquiry and participatory design, which we will consider in more detail in Chapter 13. Children are included in an *intergenerational design team* that focusses on understanding and analyzing context. Team members, including children, use a range of sketching and note-taking techniques to record their observations. Paper prototyping, using art tools familiar to children, enables both adults and children to participate in building and refining prototype designs on an equal footing. The approach has been used effectively to develop a range of new technologies for children.

As well as their likes and dislikes, children's abilities will also be different from those of adults. Younger children may have difficulty using a keyboard for instance, and may not have well-developed hand–eye coordination. Pen-based interfaces can be a useful alternative input device [300]. Again, universal design principles guide us in designing interfaces that children can use. Interfaces that allow multiple modes of input, including touch or handwriting, may be easier for children than keyboard and mouse. Redundant displays, where information is presented through text, graphics and sound will also enhance their experience.

10.4.3 Designing for cultural differences

The final area of diversity we will consider is cultural difference. Cultural difference is often used synonymously with national differences but this is too simplistic. Whilst there are clearly important national cultural differences, such as those we saw in Chapter 5, other factors such as age, gender, race, sexuality, class, religion and political persuasion, may all influence an individual's response to a system. This is particularly the case when considering websites where often the explicit intention is to design for a particular culture or subculture.

Clearly, while all of these contribute to a person's cultural identity, they will not all always be relevant in the design of a given system. However, we can draw out some key factors that we need to consider carefully if we are to practice universal design. These include language, cultural symbols, gestures and use of color.

We encountered the problem of localization of software in Chapter 5. While toolkits, with different language resource databases, facilitate the translation of menu items, error messages and other text into the local language, this does not fully deal with the language issue. Layouts and designs may reflect a language read from left to right and top to bottom, which will be unworkable with languages that do not follow this pattern.

Similarly, symbols have different meanings in different cultures. As we saw in Chapter 5, ticks ✓ and crosses ✗ represent positive and negative respectively in some cultures, and are interchangeable in others. The rainbow is a symbol of covenant with God in Judeo–Christian religions, of diversity in the gay community and of hope and peace in the cooperative movement. We cannot assume that everyone will interpret symbols in the same way and should ensure that alternative meanings of symbols will not create problems or confusion for the user. The study of the meaning of symbols is known as *semiotics* and is a worthwhile diversion for the student of universal design.

Another area where diversity can cause misunderstanding is in the use of gesture. Recently, one of the authors was teaching a new class of international students and was disconcerted to see one sitting in the front row, smiling and shaking his head. After the lecture this student came and asked a question. Every time the author asked the student if he understood the explanation, he shook his head, so further explanation ensued, much to the frustration of the student! It was only after a few minutes that it became clear: the student was from India and his gestural convention was to shake his head in agreement, the opposite of the European interpretation of the gesture. Use of gesture is quite common in video and animation and care must be taken with differences such as this. As interactions begin to incorporate gesture in virtual reality and avatars, issues such as this will become even more significant.

Finally, colors are often used in interfaces to reflect 'universal' conventions, such as red for danger and green for go. But how universal are these conventions? In fact, red and green mean many different things in different countries. As well as danger, red represents life (India), happiness (China) and royalty (France). Green is a symbol of fertility (Egypt) and youth (China) as well as safety (Anglo-American). It is difficult to assume any universal interpretation of color but the intended significance of particular colors can be supported and clarified through redundancy – providing the same information in another form as well.

10.5 SUMMARY

Universal design is about designing systems that are accessible by all users in all circumstances, taking account of human diversity in disabilities, age and culture. Universal design helps everyone – for example, designing a system so that it can be used by someone who is deaf or hard of hearing will benefit other people working in noisy environments or without audio facilities. Designing to be accessible to screen-reading systems will make websites better for mobile users and older browsers.

Multi-modal systems provide access to system information and functionality through a range of different input and output channels, exploiting redundancy. Such systems will enable users with sensory, physical or cognitive impairments to make use of the channels that they can use most effectively. But all users benefit from multi-modal systems that utilize more of our senses in an involving interactive experience.

For any design choice we should ask ourselves whether our decision is excluding someone and whether there are any potential confusions or misunderstandings in our choice.

EXERCISES

10.1 Is multi-modality always a good thing? Justify your answer.

10.2 What are (i) auditory icons and (ii) earcons? How can they be used to benefit both visually impaired and sighted users?

10.3 Research your country's legislation relating to accessibility of technology for disabled people. What are the implications of this to your future career in computing?

10.4 Take your university website or another site of your choice and assess it for accessibility using Bobby. How would you recommend improving the site?

10.5 How could systems be made more accessible to older users?

10.6 Interview either (i) a person you know over 65 or (ii) a child you know under 16 about their experience, attitude and expectations of computers. What factors would you take into account if you were designing a website aimed at this person?

10.7 Use the screen reader simulation available at www.webaim.org/simulations/screenreader to experience something of what it is like to access the web using a screen reader. Can you find the answers to the test questions on the site?

RECOMMENDED READING

M. F. Story, J. L. Mueller and R. L. Mace, *The Universal Design File: Designing for People of All Ages and Abilities*, The Center for Universal Design, NC State University, USA, 1998. Available at: www.design.ncsu.edu/cud/pubs/center/books/ud_file/toc3b14.htm (last accessed March 2003).
A full discussion of universal design principles with examples to illustrate.

A. D. N. Edwards, editor, *Extra-ordinary Human–Computer Interaction*, Cambridge University Press, 1993.
A collection of papers representing research on interfaces for users with disabilities. The first of its kind.

The Web Accessibility Initiative: www.w3.org/WAI/
The World Wide Web Consortium's own project to make the web universally accessible. Contains extensive advice and guidelines.

I. Pitt and A. Edwards, *Design of Speech-based Devices: A Practical Guide*, Springer, 2002.
Covers the use of speech in the interface, including both the practical and theoretical issues.

J. Yunker, *Beyond Borders: Web Globalization Strategies*, New Riders Publishing, 2002.
A practical guide to internationalization of websites.

D. J. Moore and J. Taylor, Interactive multimedia systems for people with autism, *Journal of Educational Media*, Vol. 25, No. 3, pp. 169–77, 2001.
A useful review of the actual and potential uses of interactive media in the education of people with autism.

PART 3

Models and theories

In all engineering disciplines, the designer recruits a selection of models to contribute to the design process. If we were building a new office block, for example, then we would use models of air circulation to design the ventilation system, structural models for the fabric and possibly social models for the detailed design of the office layout.

Models are used in other disciplines too. We may analyze the structure of a piece of music and decide that it is a rondo, or say that a poem is in sonnet form. Further, we may deliberately set out to write a sonnet, thus imposing the model upon the creative process. Craft is the art of design within constraint, and models help to formulate the constraints.

The chapters in Part 3 describe a range of models that can be used during the interface design process. Just as in the design of the office block several different types of model are required for different aspects of the building, so in interface design we would expect to use a whole selection of complementary methods.

Chapter 12 considers models with psychological or cognitive origins, where the emphasis is on formulating aspects of user behavior such as goal formation and problem solving. Chapter 13 discusses socio-technical models that attempt to describe the user within a social and organizational context, while Chapter 14 looks at models of collaboration and group interaction. Chapter 15 describes task analysis techniques for determining the relevant actions a user performs in some work domain. Chapter 16 is concerned with dialog description techniques used to specify and analyze the communication between user and system. Chapter 17 describes the use of general mathematical notations used in software engineering to specify and analyze abstract descriptions of interactive systems, and Chapter 18 extends these notations to model rich interactions.

12 Cognitive models

OVERVIEW

Cognitive models represent users of interactive systems.

- Hierarchical models represent a user's task and goal structure.
- Linguistic models represent the user–system grammar.
- Physical and device models represent human motor skills.
- Cognitive architectures underlie all of these cognitive models.

12.1 INTRODUCTION

The techniques and models in this chapter all claim to have some representation of users as they interact with an interface; that is, they model some aspect of the user's understanding, knowledge, intentions or processing. The level of representation differs from technique to technique – from models of high-level goals and the results of problem-solving activities, to descriptions of motor-level activity, such as keystrokes and mouse clicks. The formalisms have largely been developed by psychologists, or computer scientists, whose interest is in understanding user behavior.

One way to classify the models is in respect to how well they describe features of the *competence* and *performance* of the user. Quoting from Simon [323]:

> Competence models tend to be ones that can predict legal behaviour sequences but generally do this without reference to whether they could actually be executed by users. In contrast, performance models not only describe what the necessary behaviour sequences are but usually describe both what the user needs to know and how this is employed in actual task execution.

Competence models, therefore, represent the kinds of behavior expected of a user, but they provide little help in analyzing that behavior to determine its demands on the user. Performance models provide analytical power mainly by focussing on routine behavior in very limited applications.

Another useful distinction between these models is whether they address the acquisition or formulation of a plan of activity or the execution of that plan. Referring back to the interaction framework presented in Chapter 3, this classification would mean that some models are concerned with understanding the *User* and his associated task language while others are concerned with the articulation translation between that task language and the *Input* language. The presentation of the cognitive models in this chapter follows this classification scheme, divided into the following categories:

- hierarchical representation of the user's task and goal structure
- linguistic and grammatical models
- physical and device-level models.

The first category deals directly with the issue of formulation of goals and tasks. The second deals with the grammar of the articulation translation and how it is understood by the user. The third category again deals with articulation, but at the human motor level instead of at a higher level of human understanding.

Architectural assumptions about the user are needed in any of the cognitive models discussed here. Some of the more basic architectural assumptions were covered in Chapter 1, such as the distinction between long- and short-term memory. After discussing models in the three categories above, we will describe two additional cognitive architectures and how they are relevant for analyzing interactive system design.

Many of these nominally cognitive models have a rather computational flavor. This reflects the way that computational analogies are often used in cognitive psychology. The similarity between the language describing the user and that describing the computer has some advantages and some dangers. On the positive side it makes communication and analysis of the combined human–computer system easier. For instance, cognitive complexity theory (described later) produces models of both user goals and the system grammar, and can reason about their interaction. On the other hand, there is a danger that this will encourage a mechanistic view of the user.

12.2 GOAL AND TASK HIERARCHIES

Many models make use of a model of mental processing in which the user achieves goals by solving subgoals in a divide-and-conquer fashion. We will consider two models, *GOMS* and *CCT*, where this is a central feature. However, we will see similar features in other models, such as *TAG* (Section 12.3.2) and when we consider task analysis techniques (Chapter 15).

Imagine we want to produce a report on sales of introductory HCI textbooks. To achieve this goal we divide it into several subgoals, say gathering the data together, producing the tables and histograms, and writing the descriptive material. Concentrating on the data gathering, we decide to split this into further subgoals: find the names of all introductory HCI textbooks and then search the book sales database for these books. Similarly, each of the other subgoals is divided up into further subgoals, until some level of detail is found at which we decide to stop. We thus end up with a hierarchy of goals and subgoals. The example can be laid out to expose this structure:

```
produce report
    gather data
    .   find book names
    .   .   do keywords search of names database
    .   .       <<further subgoals>>
    .   .   sift through names and abstracts by hand
    .   .       <<further subgoals>>
    .   search sales database
    .       <<further subgoals>>
    layout tables and histograms
        <<further subgoals>>
    write description
        <<further subgoals>>
```

Various issues arise as one attempts such analyses of computer use.

Where do we stop? We can go on decomposing tasks until we get down to the individual hand and eye movements of the user, or we can stop at a more abstract level. Where do we start? In a similar way, we can start our analyses at different points in the hierarchy of goals. At the extreme we could extend our analysis to larger and larger goals: 'light hob' is a subgoal of 'boil peas' and so on to goals such as 'have my dinner', 'feed' and 'stay alive'.

These two questions are issues of *granularity*, and both of the methods described below leave this to some extent in the hands of the designer. Different design issues demand different levels of analysis. However, both methods operate at a relatively low level; neither would attempt to start with such an abstract goal as 'produce a report' which will involve real creativity and difficult problem solving. Instead they confine themselves to more routine learned behavior. This most abstract task is referred to as the *unit task*. The unit task does not require any problem-solving skills on the part of the user, though it frequently demands quite sophisticated problem-solving skills on the part of the designer to determine them.

What do we do when there are several ways of solving a problem, or if the solutions to two subgoals interact? Users will often have more than one way to achieve a goal and there must be some way of representing how they select between competing solutions.

Another important issue has to do with the treatment of error. Users are not perfect. A goal hierarchy may show how the perfect user would achieve a goal, but what can it say about difficulties the user may have along the way? In general, prediction of error behavior is poor amongst these hierarchical modeling techniques, though some (cognitive complexity theory (CCT), for example) can represent error behavior.

12.2.1 GOMS

The *GOMS* model of Card, Moran and Newell is an acronym for Goals, Operators, Methods and Selection [56]. A GOMS description consists of these four elements:

Goals These are the user's goals, describing what the user wants to achieve. Further, in GOMS the goals are taken to represent a 'memory point' for the user, from which he can evaluate what should be done and to which he may return should any errors occur.

Operators These are the lowest level of analysis. They are the basic actions that the user must perform in order to use the system. They may affect the system (for example, press the 'X' key) or only the user's mental state (for example, read the dialog box). There is still a degree of flexibility about the granularity of operators; we may take the command level 'issue the SELECT command' or be more primitive: 'move mouse to menu bar, press center mouse button . . .'.

Methods As we have already noted, there are typically several ways in which a goal can be split into subgoals. For instance, in a certain window manager a currently selected window can be closed to an icon either by selecting the 'CLOSE' option from a pop-up menu, or by hitting the 'L7' function key. In GOMS these two goal decompositions are referred to as methods, so we have the `CLOSE-METHOD` and the `L7-METHOD`:

```
GOAL: ICONIZE-WINDOW
.     [select GOAL: USE-CLOSE-METHOD
.               .      MOVE-MOUSE-TO-WINDOW-HEADER
.               .      POP-UP-MENU
.               .      CLICK-OVER-CLOSE-OPTION
                GOAL: USE-L7-METHOD
.               .      PRESS-L7-KEY]
```

The dots are used to indicate the hierarchical level of goals.

Selection From the above snippet we see the use of the word `select` where the choice of methods arises. GOMS does not leave this as a random choice, but attempts to predict which methods will be used. This typically depends both on the particular user and on the state of the system and details about the goals. For instance, a user, Sam, never uses the `L7-METHOD`, except for one game, ‘`blocks`’, where the mouse needs to be used in the game until the very moment the key is pressed. GOMS captures this in a selection rule for Sam:

User Sam:
`Rule 1:` Use the `CLOSE-METHOD` unless another rule applies.
`Rule 2:` If the application is ‘`blocks`’ use the `L7-METHOD`.

The goal hierarchies described in a GOMS analysis are almost wholly below the level of the unit task defined earlier. A typical GOMS analysis would therefore consist of a single high-level goal, which is then decomposed into a sequence of unit tasks, all of which can be further decomposed down to the level of basic operators:

```
GOAL: EDIT-MANUSCRIPT
.   GOAL: EDIT-UNIT-TASK repeat until no more unit tasks
```

The goal decomposition between the overall task and the unit tasks would involve detailed understanding of the user’s problem-solving strategies and of the application domain. These are side-stepped entirely by the method as originally proposed. It would be possible to use the general notation in order to describe this subgoal structure (as for instance in the book report example above). This form of high-level goal description is adopted during *task analysis* which will be discussed in Chapter 15. In particular, the aim of *hierarchical task analysis* is to produce task decompositions, which would be similar (but in a different notation) to that in the book report example.

Analysis of the GOMS goal structure can yield measures of performance. The stacking depth of a goal structure can be used to estimate short-term memory requirements. The model of the users’ mental processes implied by this is, of course, very idealized. Also, the selection rules can be tested for accuracy against user traces, and changed in response to discrepancies. In early experiments on the technique, the inventors were able to achieve on average a 90% correct prediction rate of user commands. Further, a very simple method of predicting times (basically assuming that each operator takes a constant time) was able to predict actual times with an error of 33%.

The original GOMS model has served as the basis for much of the cognitive modeling research in HCI. It was good for describing how experts perform routine tasks. Coupled with the physical device models discussed later, it can be used to predict the performance of these users in terms of execution times. It was never intended to provide the kind of information about the user's knowledge that could be compared across different tasks in order to predict things like training or transfer times.

DESIGN FOCUS

GOMS saves money

Some years ago the US telephone company NYNEX were intending to install a new computer system to support their operators. Before installation a detailed GOMS analysis was performed taking into account the cognitive and physical processes involved in dealing with a call. The particular technique was rather different from the original GOMS notation as described here. Because an operator performs several activities in parallel a PERT-style GOMS description was constructed [192, 154]. The PERT analysis was used to determine the critical path, and hence the time to complete a typical task. It was discovered that rather than speeding up operations, the new system would take longer to process each call. The new system was abandoned before installation, leading to a saving of many millions of dollars.

Worked exercise *Create a GOMS description of the task of photocopying an article from a journal. Discuss the issue of closure (see Chapter 1) in terms of your GOMS description.*

Answer One possible GOMS description of the goal hierarchy for this task is given below. Answers will vary depending on assumptions about the photocopier used as the model for the exercise. In this example, we will assume that the article is to be copied one page at a time and that a cover over the imaging surface of the copier has to be in place before the actual copy can be made.

```
GOAL: PHOTOCOPY-PAPER
.     GOAL: LOCATE-ARTICLE
.     GOAL: PHOTOCOPY-PAGE repeat until no more pages
.     .     GOAL: ORIENT-PAGE
.     .     .       OPEN-COVER
.     .     .       SELECT-PAGE
.     .     .       POSITION-PAGE
.     .     .       CLOSE-COVER
.     .     GOAL: PRESS-COPY-BUTTON
.     .     GOAL: VERIFY-COPY
.     .     .       LOCATE-OUT-TRAY
.     .     .       EXAMINE-COPY
.     GOAL: COLLECT-COPY
.     .     LOCATE-OUT-TRAY
.       .       REMOVE-COPY (outer goal satisfied!)
.       GOAL: RETRIEVE-JOURNAL
.       .       OPEN-COVER
.       .       REMOVE-JOURNAL
.       .       CLOSE-COVER
```

The closure problem which appears in this example occurs when the copy of the article is removed from the photocopier out tray, satisfying the overall goal for the task. In the above description, however, the original journal article is still on the imaging surface of the photocopier, and the cover is closed. The user could easily forget to remove the journal. How could the photocopying procedure be revised to eliminate this problem? One answer is to force the goal `RETRIEVE-JOURNAL` to be satisfied before `COLLECT-COPY`.

12.2.2 Cognitive complexity theory

Cognitive complexity theory, introduced by Kieras and Polson [199], begins with the basic premises of goal decomposition from GOMS and enriches the model to provide more predictive power. CCT has two parallel descriptions: one of the user's goals and the other of the computer system (called the *device* in CCT). The description of the user's goals is based on a GOMS-like goal hierarchy, but is expressed primarily using *production rules*. We introduced production rules in Chapter 1 and we further describe their use in CCT below. For the system grammar, CCT uses *generalized transition networks*, a form of *state transition network*. This will not be described here, but state transition networks will be discussed in detail in Chapter 16.

The production rules are a sequence of rules:

if *condition* then *action*

where *condition* is a statement about the contents of working memory. If the condition is true then the production rule is said to fire. An *action* may consist of one or more elementary actions, which may be either changes to the working memory, or external actions such as keystrokes. The production rule 'program' is written in a LISP-like language.

As an example, we consider an editing task using the UNIX vi text editor. The task is to insert a space where one has been missed out in the text, for instance if we noticed that in the above paragraph we had written 'cognitivecomplexity theory'. This is a reasonably frequent typing error and so we assume that we have developed good procedures to perform the task. We consider a fragment of the associated CCT production rules.

```
(SELECT-INSERT-SPACE
IF (AND (TEST-GOAL perform unit task)
      (TEST-TEXT task is insert space)
      (NOT (TEST-GOAL insert space))
      (NOT (TEST-NOTE executing insert space)) )
THEN ( (ADD-GOAL insert space)
       (ADD-NOTE executing insert space)
       (LOOK-TEXT task is at %LINE %COL) ))
(INSERT-SPACE-DONE
IF (AND (TEST-GOAL perform unit task)
       (TEST-NOTE executing insert space)
       (NOT (TEST-GOAL insert space)) )
THEN ( (DELETE-NOTE executing insert space)
       (DELETE-GOAL perform unit task)
       (UNBIND %LINE %COL) ))
(INSERT-SPACE-1
IF (AND (TEST-GOAL insert space)
       (NOT (TEST-GOAL move cursor))
       (NOT (TEST-CURSOR %LINE %COL)) )
THEN ( (ADD-GOAL move cursor to %LINE %COL) ))
(INSERT-SPACE-2
IF (AND (TEST-GOAL insert space)
       (TEST-CURSOR %LINE %COL) )
THEN ( (DO-KEYSTROKE 'I')
       (DO-KEYSTROKE SPACE)
       (DO-KEYSTROKE ESC)
       (DELETE-GOAL insert space) ))
```

To see how these rules work, imagine that the user has just seen the typing mistake and thus the contents of working memory (w.m.) are

```
(GOAL perform unit task)
(TEXT task is insert space)
(TEXT task is at 5 23)
(CURSOR 8 7)
```

`TEXT` refers to the text of the manuscript that is being edited and `CURSOR` refers to the insertion cursor on the screen. Of course, these items are not actually located in working memory – they are external to the user – but we assume that knowledge from observing them is stored in the user's working memory.

The location (5,23) is the line and column of the typing mistake where the space is required. However, the current cursor position is at line 8 and column 7. This is of course acquired into the user's working memory by looking at the screen. Looking at the four rules above (`SELECT-INSERT-SPACE`, `INSERT-SPACE-DONE`, `INSERT-SPACE-1` and `INSERT-SPACE-2`), only the first can fire. The condition for `SELECT-INSERT-SPACE` is:

```
(AND (TEST-GOAL perform unit task)
             true because (GOAL perform unit task) is in w.m.
     (TEST-TEXT task is insert space)
             true because (TEXT task is insert space) is in w.m.
(NOT (TEST-GOAL insert space))
        true because (GOAL insert space) is not in w.m.
(NOT (TEST-NOTE executing insert space)) )
        true because (NOTE executing insert space)
        is not in w.m.
```

So, the rule fires and its action is performed. This action has no external effect in terms of keystrokes, but adds extra information to working memory. The (LOOK-TEXT task is at %LINE %COL) looks for a corresponding entry and *binds* LINE and COL to 5 and 23 respectively. These are variables, somewhat as in a normal programming language, which are referred to again in other rules.

The contents of working memory after the firing of rule SELECT-INSERT-SPACE are as follows (note that the order of elements of working memory is arbitrary):

```
(GOAL perform unit task)
(TEXT task is insert space)
(TEXT task is at 5 23)
(NOTE executing insert space)
(GOAL insert space)
(LINE 5)
(COL 23)
(CURSOR 8 7)
```

At this point neither rule SELECT-INSERT-SPACE nor INSERT-SPACE-DONE will fire as the entry (GOAL insert space) will make their conditions false. As LINE is bound to 5 and COL is bound to 23, the condition (TEST-CURSOR %LINE %COL) will be false also, and hence only rule INSERT-SPACE-1 can fire.

After this rule's actions have been performed, the working memory will include the entry (GOAL move cursor to 5 23). The rules for moving the cursor are not included here, but would be quite extensive, moving up/down and right/left depending on the relative positions of the cursor and the target location. Eventually, assuming the cursor movement is successful, the cursor would be at (5,23) whence rule INSERT-SPACE-2 would be able to fire. This would perform the keystrokes: I, SPACE and ESC, which in vi puts the editor into insert mode, types the space and then leaves insert mode. The action also removes the insert space goal from working memory as this goal has been achieved.

Now the goal has been removed, the second rule INSERT-SPACE-DONE is free to fire, which 'tidies up' working memory. In particular, it 'unbinds' the variables LINE and COL, that is it removes the bindings for them from working memory.

Notice that the rules did not fire in the order they were written. Although they look somewhat like the if–then–else commands one would get in a standard programming language, they behave very differently. The rules are all active and at each moment any rule that has its conditions true may fire. Some rules may never fire; for instance, if the cursor is at the correct position the third rule would not fire. Furthermore, the same rule may fire repeatedly; for example, if we were to write out the production rules for moving the cursor, one rule may well be

```
(MOVE-UP
IF (AND (TEST-GOAL move-up)
     (TEST-CURSOR-BELOW %LINE) )
THEN ( (     DO-KEYSTROKE 'K') ))
```

This rule is to type 'K' (the vi command to move the cursor up one line) while the cursor is below the desired line. It will, of course, be constantly refired until the cursor is at the correct line.

Notice that the keystrokes for actually inserting the space, once you are at the right position, have been *proceduralized*. That is, the user does not go through the subgoals 'enter insert mode', 'type space', 'leave insert mode'. For a complex insertion, it is quite likely that the user will perform exactly these goals. However, the act of inserting a single space is assumed to be so well rehearsed that it is stored as a single chunk. That is, the rules above represent *expert* knowledge of the vi editor.

Of course, novices may well do exactly the same keystrokes as the experts, but the way they store the knowledge will be different. To cope with this, CCT has a set of 'style' rules for novices. These limit the form of the conditions and actions in the production rules. Basically, novices are expected to test constantly all the rules in their working memory and to check for feedback from the system after every keystroke. Thus a set of 'novice' rules would not include the proceduralized form of insert space. Bovair, Kieras and Polson provide a list of many style rules which can be used to embody certain psychological assumptions about the user (novice/expert distinction is only one) in a CCT description [39].

The rules in CCT need not represent error-free performance. They can be used to explain error phenomena, though they cannot predict them. For instance, the rules above for inserting a space are 'buggy' – they do not check the editor's mode. Imagine you had just been typing the 'cognitive' in 'cognitivecomplexity theory' (with the space missing), you think for a few minutes and then look again at the screen and notice that the space is missing. The cursor is at the correct position for the space, so rule `INSERT-SPACE-1` never gets fired and we go directly through the sequence: `SELECT-INSERT-SPACE`, `INSERT-SPACE-2` then `INSERT-SPACE-DONE`. You type 'i', a space and then escape. However, the 'i' assumes that you are in vi's command mode, and is the command to move the editor into insert mode. If, however, after typing 'cognitive' you had not typed escape, to get you back into command mode, the whole sequence would be done in insert mode. The text would read: 'cognitiveI complexity theory'.

The CCT rules are closely related to GOMS-like goal hierarchies; the rules may be generated from such a hierarchy, or alternatively, we may analyze the production rules to obtain the goal tree:

```
GOAL: insert space
.     GOAL: move cursor - if not at right position
.     PRESS-KEY-I
.     PRESS-SPACE
.     PRESS-ESCAPE
```

The stacking depth of this goal hierarchy (as described for GOMS) is directly related to the number of (`GOAL ...`) terms in working memory.

In fact, the CCT rules can represent more complex plans than the simple sequential hierarchies of GOMS. The continuous activity of all production rules makes it possible to represent concurrent plans. For example, one could have one set of production rules representing the goal of writing a book, and another set representing the goal of drinking tea. These rules could both be active simultaneously, thus allowing an author to drink tea whilst typing. Despite this apparent flexibility, CCT is not normally used in this way. It is not clear why this is, except that CCT, like GOMS, is aimed at low-level, *proceduralized* goals – that is, the *unit task*. It is reasonable that successive unit tasks be chosen from different activities: the author may delete a word, have a drink, do a word search, but each time a complete unit task would be performed – the author does not take a drink of tea in the middle of deleting a word.

We have seen how CCT rules may be informally analyzed to discuss issues of proceduralization and error behavior, and how we can relate them to GOMS-like goal hierarchies. However, the main aim of CCT is (as its name suggests) to be able to measure the complexity of an interface.

Basically, the more production rules in the CCT description the more difficult the interface is to learn. This claim rests on the assumption that the production rules represent reasonably accurately the way knowledge is stored and therefore that the time taken to learn an interface is roughly proportional to the number of rules you have to learn.

We have only discussed the user side of CCT here. If the cognitive user description is complemented by a description of the system, it is claimed that one can predict the difficulty of the mapping between the user's goals and the system model. The generalized transition networks that describe the system grammar themselves have a hierarchical structure. Thus both the description of the user and that of the system can be represented as hierarchies. These can then be compared to find mismatches and to produce a measure of dissonance.

There are various problems with CCT. As with many 'rich' description methods, the size of description for even a part of an interface can be enormous. Furthermore, there may be several ways of representing the same user behavior and interface behavior, yielding different measures of dissonance. To some extent this is catered for by the novice style rules, but there is no such set of rules for the system description.

Another problem is the particular choice of notations. Production rules are often suggested as a good model of the way people remember procedural knowledge, but there are obvious 'cludges' in the CCT description given above. In particular, the working memory entry (`NOTE executing insert space`) is there purely to allow the `INSERT-SPACE-DONE` rule to fire at the appropriate time. It is not at all clear that it has any real cognitive significance. One may also question whether the particular notation chosen for the system is critical to the method. One might choose to represent the system using any one of the dialog description notations in Chapter 16. Different notations would probably yield slightly different measures of dissonance.

However, one should regard CCT as an engineering tool giving one a rough measure of learnability and difficulty combined with a detailed description of user behavior.

This can then be used by analysts employing their professional expertise. Arguably, the strength of the central idea of CCT lies beyond the particular notations used.

12.2.3 Problems and extensions of goal hierarchies

The formation of a goal hierarchy is largely a post hoc technique and runs a very real risk of being defined by the computer dialog rather than the user. One way to rectify this is to produce a goal structure based on pre-existing manual procedures and thus obtain a natural hierarchy [201]. To be fair, GOMS defines its domain to be that of expert use, and thus the goal structures that are important are those which users develop out of their use of the system. However, such a natural hierarchy may be particularly useful as part of a CCT analysis, representing a very early state of knowledge.

On the positive side, the conceptual framework of goal hierarchies and user goal stacks can be used to express interface issues, not directly addressed by the notations above. For instance, we can use this to examine in more detail the closure problem with early automated teller machines (ATMs) mentioned in the Design Focus box in Chapter 1, Section 1.3.2. These early ATMs gave the customers the money before returning their cards. Unfortunately, this led to many customers leaving their cards behind. This was despite on-screen messages telling them to wait. This is referred to as a problem of *closure*. The user's principal goal is to get money; when that goal is satisfied, the user does not complete or close the various subtasks which still remain open:

```
GOAL: GET-MONEY
.   GOAL: USE-ATM
.   .   INSERT-CARD
.   .   ENTER-PIN
.   .   ENTER-AMOUNT
.   .   COLLECT-MONEY
            << outer goal now satisfied goal stack popped >>
.   .   COLLECT-CARD - subgoal operators missed
```

Banks (at least some of them) soon changed the dialog order so that the card was always retrieved before the money was dispensed. A general rule that can be applied to any goal hierarchy from this is that no higher-level goal should be satisfied until all subgoals have been satisfied. However, it is not always easy to predict when the user will consider a goal to have been satisfied. For instance, one of the authors has been known to collect his card and forget the money!

12.3 LINGUISTIC MODELS

The user's interaction with a computer is often viewed in terms of a language, so it is not surprising that several modeling formalisms have developed centered around this concept. Several of the dialog notations described in Chapter 16 are also based on linguistic ideas. Indeed, BNF grammars are frequently used to specify dialogs. The models here, although similar in form to dialog design notations, have been proposed with the intention of understanding the user's behavior and analyzing the cognitive difficulty of the interface.

12.3.1 BNF

Representative of the *linguistic approach* is Reisner's use of Backus–Naur Form (*BNF*) rules to describe the dialog grammar [301]. This views the dialog at a purely syntactic level, ignoring the semantics of the language. BNF has been used widely to specify the syntax of computer programming languages, and many system dialogs can be described easily using BNF rules. For example, imagine a graphics system that has a line-drawing function. To select the function the user must select the 'line' menu option. The line-drawing function allows the user to draw a polyline, that is a sequence of line arcs between points. The user selects the points by clicking the mouse button in the drawing area. The user double clicks to indicate the last point of the polyline.

```
draw-line       ::= select-line + choose-points
                    + last-point
select-line     ::= position-mouse + CLICK-MOUSE
choose-points   ::= choose-one
                    | choose-one + choose-points
choose-one      ::= position-mouse + CLICK-MOUSE
last-point      ::= position-mouse + DOUBLE-CLICK-MOUSE
position-mouse  ::= empty | MOVE-MOUSE + position-mouse
```

The names in the description are of two types: *non-terminals*, shown in lower case, and *terminals*, shown in upper case. Terminals represent the lowest level of user behavior, such as pressing a key, clicking a mouse button or moving the mouse. Non-terminals are higher-level abstractions. The non-terminals are defined in terms of other non-terminals and terminals by a definition of the form

```
name ::= expression
```

The '`::=`' symbol is read as 'is defined as'. Only non-terminals may appear on the left of a definition. The right-hand side is built up using two operators '`+`' (sequence) and '`|`' (choice). For example, the first rule says that the non-terminal `draw-line` is defined to be `select-line` followed by `choose-points` followed by `last-point`. All of these are non-terminals, that is they do not tell us what the basic user actions are. The second rule says that `select-line` is defined to be `position-mouse` (intended to be over the 'line' menu entry) followed by `CLICK-MOUSE`. This is our first terminal and represents the actual clicking of a mouse.

To see what `position-mouse` is, we look at the last rule. This tells us that there are two possibilities for `position-mouse` (separated by the '`|`' symbol). One option is that `position-mouse` is `empty` – a special symbol representing no action. That is, one option is not to move the mouse at all. The other option is to do a `MOVE-MOUSE` action followed by `position-mouse`. This rule is recursive, and this second `position-mouse` may itself either be empty or be a `MOVE-MOUSE` action followed by `position-mouse`, and so on. That is, `position-mouse` may be any number of `MOVE-MOUSE` actions whatsoever.

Similarly, `choose-points` is defined recursively, but this time it does not have the option of being empty. It may be *one or more* of the non-terminal `choose-one` which is itself defined to be (like `select-line`) `position-mouse` followed by `CLICK-MOUSE`.

The BNF description of an interface can be analyzed in various ways. One measure is to count the number of rules. The more rules an interface requires to use it, the more complicated it is. This measure is rather sensitive to the exact way the interface is described. For example, we could have replaced the rules for `choose-points` and `choose-one` with the single definition

```
choose-points ::= position-mouse + CLICK-MOUSE
              | position-mouse + CLICK-MOUSE + choose-points
```

A more robust measure also counts the number of '+' and '|' operators. This would, in effect, penalize the more complex single rule. Another problem arises with the rule for `select-line`. This is identical to the `choose-one` rule. However, the acts of selecting a menu option and choosing a point on a drawing surface are obviously so different that they must surely be separated. Decisions like this about the structure of a BNF description are less of a problem in practice than the corresponding problems we had with CCT.

In addition to these complexity measures for the language as a whole, we can use the BNF definition to work out how many basic actions are required for a particular task, and thus obtain a crude estimate of the difficulty of that task.

The BNF description above only represented the user's actions, not the user's perception of the system's responses. This input bias is surprisingly common amongst cognitive models, as we will discuss in Section 12.4. Reisner has developed extensions to the basic BNF descriptions, which attempt to deal with this by adding 'information-seeking actions' to the grammar.

12.3.2 Task–action grammar

Measures based upon BNF have been criticized as not 'cognitive' enough. They ignore the advantages of consistency both in the language's structure and in its use of command names and letters. *Task–action grammar* (*TAG*) [284] attempts to deal with some of these problems by including elements such as parametrized grammar rules to emphasize consistency and encoding the user's world knowledge (for example, up is the opposite of down).

To illustrate consistency, we consider the three UNIX commands: `cp` (for copying files), `mv` (for moving files) and `ln` (for linking files). Each of these has two possible forms. They either have two arguments, a source and destination filename, or have any number of source filenames followed by a destination directory:

```
copy ::=   'cp' + filename + filename
         | 'cp' + filenames + directory
move ::=   'mv' + filename + filename
         | 'mv' + filenames + directory
link ::=   'ln' + filename + filename
         | 'ln' + filenames + directory
```

Measures based upon BNF could not distinguish between these consistent commands and an inconsistent alternative – say if `ln` took its directory argument first. Task–action grammar was designed to reveal just this sort of consistency. Its description of the UNIX commands would be

```
file-op[Op] :=   command[Op] + filename + filename
               | command[Op] + filenames + directory
command[Op=copy] := 'cp'
command[Op=move] := 'mv'
command[Op=link] := 'ln'
```

This captures the consistency of the commands and closely resembles the original textual description. One would imagine that a measure of the complexity of the language based on the TAG description would be better at predicting actual learning and performance than a simple BNF one.

As well as handling consistency well, TAG has features for talking about 'world knowledge'. For example, imagine we have two command line interfaces for moving a mechanical turtle around the floor.

Command interface 1

```
movement[Direction]
      := command[Direction] + distance + RETURN
command[Direction=forward]   := 'go 395'
command[Direction=backward]  := 'go 013'
command[Direction=left]      := 'go 712'
command[Direction=right]     := 'go 956'
```

Command interface 2

```
movement[Direction]
      := command[Direction] + distance + RETURN
command[Direction=forward]   := 'FORWARD'
command[Direction=backward]  := 'BACKWARD'
command[Direction=left]      := 'LEFT'
command[Direction=right]     := 'RIGHT'
```

The first interface may not be as silly as it seems; the command `'go 395'` could refer to the address of a machine-code routine, which performs the appropriate movement. However, it is absolutely clear that the second interface is preferable to the first. TAG includes a special form `known-item`, which is used to denote information that the user will already know, and thus not need to learn in order to use the system. Using this form, the TAG rules for the second interface are rewritten

Command interface 2

```
movement[Direction]
      := command[Direction] + distance + RETURN
command[Direction]:= known-item[Type=word,Direction]
* command[Direction=forward]  := 'FORWARD'
* command[Direction=backward] := 'BACKWARD'
* command[Direction=left]     := 'LEFT'
* command[Direction=right]    := 'RIGHT'
```

The starred rules can be generated from the second rule using the user's world knowledge. They are included in any TAG description for completeness, but are not counted in any measure of complexity.

Sometimes it may not be clear what the appropriate command is, but once we know one, the rest become obvious. For example, consider a simple database displaying a list of records. We are expecting two commands, one to move up the list to the previous record, and another to move down the list to the next record. There are several options for the commands, for instance UP/DOWN, PREVIOUS/NEXT, possibly in upper or lower case, possibly also just the first letter of the relevant word. In addition, one might have mixed-up command sets such as UP/NEXT or N/previous. The fact that any of the former set of commands is easier to learn than the mixed-up commands is called *congruence*. TAG has a notation to describe the congruence of an interface. The notation `F('next')` is used to denote the feature set related to the word 'next'. That is, next/previous. With this notation a congruent grammar requires only one 'real' rule, such as

```
browse[Direction] := F('next') + return
* browse[Direction=up]   := 'previous' + return
* browse[Direction=down] := 'next' + return
```

We have seen that the notation allows one to say that the commands RIGHT and LEFT are consistent for opposite actions. How do we know that the user regards the opposite of RIGHT to be LEFT rather than WRONG? Obviously, the inclusion of world knowledge depends upon the user of the system – the above certainly assumes that the user's language is English. The designer is responsible for the input of this world knowledge into the TAG description and its validity will depend on the professional judgment of the designer. However, TAG will make these assumptions clear and thus, by highlighting them, hold them up for inspection.

12.4 THE CHALLENGE OF DISPLAY-BASED SYSTEMS

Both goal hierarchical and grammar-based techniques were initially developed when most interactive systems were command line, or at most, keyboard and cursor based. There are significant worries, therefore, about how well these approaches can generalize to deal with more modern windowed and mouse-driven interfaces.

Both families of techniques largely ignore system output – what the user sees. The implicit assumption is that the users know exactly what they want to do and execute the appropriate command sequences blindly. There are exceptions to this. We have already mentioned how Reisner's BNF has been extended to include assertions about output. In addition, TAG has been extended to include information about how the display can affect the grammar rules [180].

Another problem for grammars is the lowest-level lexical structure. Pressing a cursor key is a reasonable *lexeme*, but moving a mouse one pixel is less sensible. In addition, pointer-based dialogs are more display oriented. Clicking a cursor at a particular point on the screen has a meaning dependent on the current screen contents. This problem can be partially resolved by regarding operations such as 'select region of text' or 'click on quit button' as the terminals of the grammar. If this approach is taken, the detailed mouse movements and parsing of mouse events in the context of display information (menus, etc.) are abstracted away.

Goal hierarchy methods have different problems, as more display-oriented systems encourage less structured methods for goal achievement. Instead of having well-defined plans, the user is seen as performing a more exploratory task, recognizing fruitful directions and backing out of others. Typically, even when this exploratory style is used at one level, we can see within it and around it more goal-oriented methods. So, for example, we might consider the high-level goal structure

```
WRITE_LETTER
.      FIND_SIMILAR_LETTER
.      COPY_IT
.      EDIT_COPY
```

However, the task of finding a similar letter would be exploratory, searching through folders, etc. Such recognition-based searching is extremely difficult to represent as a goal structure. Similarly, the actual editing would depend very much on non-planned activities: 'ah yes, I want to reuse that bit, but I'll have to change that'. If we then drop to a lower level again, goal hierarchies become more applicable. For instance, during the editing stage we might have the 'delete a word' subdialog:

```
DELETE_WORD
.      SELECT_WORD
.      .        MOVE_MOUSE_TO_WORD_START
.      .        DEPRESS_MOUSE_BUTTON
.      .        MOVE_MOUSE_TO_WORD_END
.      .        RELEASE_MOUSE_BUTTON
.      CLICK_ON_DELETE
.      .        MOVE_MOUSE_TO_DELETE_ICON
.      .        CLICK_MOUSE_BUTTON
```

Thus goal hierarchies can partially cope with display-oriented systems by an appropriate choice of level, but the problems do emphasize the rather prescriptive nature of the cognitive models underlying them.

These problems have been one of the factors behind the growing popularity of *situated action* [334] and *distributed cognition* [208, 185] in HCI (see also Chapter 14). Both approaches emphasize the way in which actions are contingent upon events and determined by context, rather than being pre-planned. At one extreme, protagonists of these approaches seem to deny any planned actions or long-term goals. On the other hand, traditional cognitive modelers are modeling *display-based cognition* using production rules and similar methods, which include sensory data within the models.

At a low level, chunked expert behavior is modeled effectively using hierarchical or linguistic models, and is where the *keystroke-level model* (discussed in Section 12.5.1 below) has proved effective. In contrast, it is clear that no amount of cognitive modeling can capture the activity during the writing of a poem. Between these two, cognitive models will have differing levels of success and utility. Certainly, models at all but the lowest levels must take into account the user's reactions to feedback from the system, otherwise they cannnot address the fundamental issue of *interactivity* at all.

12.5 PHYSICAL AND DEVICE MODELS

12.5.1 Keystroke-level model

Compared with the deep cognitive understanding required to describe problem-solving activities, the human motor system is well understood. *KLM* (*Keystroke-Level Model* [55]) uses this understanding as a basis for detailed predictions about user performance. It is aimed at unit tasks within interaction – the execution of simple command sequences, typically taking no more than 20 seconds. Examples of this would be using a search and replace feature, or changing the font of a word. It does not extend to complex actions such as producing a diagram. The assumption is that these more complex tasks would be split into subtasks (as in GOMS) before the user attempts to map them into physical actions. The task is split into two phases:

acquisition of the task, when the user builds a mental representation of the task;

execution of the task using the system's facilities.

KLM only gives predictions for the latter stage of activity. During the acquisition phase, the user will have decided how to accomplish the task using the primitives of the system, and thus, during the execution phase, there is no high-level mental activity – the user is effectively expert. KLM is related to the GOMS model, and can be thought of as a very low-level GOMS model where the method is given.

The model decomposes the execution phase into five different physical motor operators, a mental operator and a system response operator:

K	Keystroking, actually striking keys, including shifts and other modifier keys.
B	Pressing a mouse button.
P	Pointing, moving the mouse (or similar device) at a target.

H Homing, switching the hand between mouse and keyboard.

D Drawing lines using the mouse.

M Mentally preparing for a physical action.

R System response which may be ignored if the user does not have to wait for it, as in copy typing.

The execution of a task will involve interleaved occurrences of the various operators. For instance, imagine we are using a mouse-based editor. If we notice a single character error we will point at the error, delete the character and retype it, and then return to our previous typing point. This is decomposed as follows:

1. Move hand to mouse	**H**[mouse]
2. Position mouse after bad character	**PB**[LEFT]
3. Return to keyboard	**H**[keyboard]
4. Delete character	**MK**[DELETE]
5. Type correction	**K**[char]
6. Reposition insertion point	**H**[mouse]**MPB**[LEFT]

Notice that some operators have descriptions added to them, representing which device the hand homes to (for example, [mouse]) and what keys are hit (for example, LEFT – the left mouse button).

The model predicts the total time taken during the execution phase by adding the component times for each of the above activities. For example, if the time taken for one keystroke is t_K, then the total time doing keystrokes is

$$T_K = 2t_K$$

Similar calculations for the rest of the operators give a total time of

$$\begin{aligned} T_{\text{execute}} &= T_K + T_B + T_P + T_H + T_D + T_M + T_R \\ &= 2t_K + 2t_B + t_P + 3t_H + 0 + 2t_M + 0 \end{aligned}$$

In this example, the system response time was zero. However, if the user had to wait for the system then the appropriate time would be added. In many typing tasks, the user can type ahead anyway and thus there is no need to add response times. Where needed, the response time can be measured by observing the system.

The times for the other operators are obtained from empirical data. The keying time obviously depends on the typing skill of the user, and different times are thus used for different users. Pressing a mouse button is usually quicker than typing (especially for two-finger typists), and a more accurate time prediction can be made by separating out the button presses **B** from the rest of the keystrokes **K**. The pointing time can be calculated using Fitts' law (see Chapter 1), and thus depends on the size and position of the target.[1] Alternatively, a fixed time based on average within

1 The form of Fitts' law used with the original KLM is $K \log_2(D/S + 0.5)$, where D is the distance to the target and S is the target size. We will use this form for calculations in this subsection, but revert to the form $a + b \log_2(D/S + 1)$ in the next subsection when we consider Buxton's three-state model, as this form was used for these experiments.

Table 12.1 Times for various operators in the keystroke-level model (adapted from Card, Moran and Newell [56], published and reprinted by permission of Lawrence Erlbaum Associates, Inc.)

Operator	Remarks	Time (s)
K	Press key	
	good typist (90 wpm)	0.12
	poor typist (40 wpm)	0.28
	non-typist	1.20
B	Mouse button press	
	down or up	0.10
	click	0.20
P	Point with mouse	
	Fitts' law	$0.1 \log_2(D/S + 0.5)$
	average movement	1.10
H	Home hands to and from keyboard	0.40
D	Drawing – domain dependent	–
M	Mentally prepare	1.35
R	Response from system – measure	–

wpm = words per minute

screen pointing can be used. Drawing time depends on the number and length of the lines drawn, and is fairly domain specific, but one can easily use empirical data for more general drawing tasks. Finally, homing time and mental preparation time are assumed constant. Typical times are summarized in Table 12.1.

The mental operator is probably the most complex part of KLM. Remember that the user is assumed to have decided what to do, and how to do it. The mental preparation is thus just the slight pauses made as the user recalls what to do next. There are complicated heuristics for deciding where to put **M** operators, but they all boil down to the level of chunking (see Chapter 1 for a discussion of chunking). If the user types a word, or a well-known command name, this will be one chunk, and hence only require one mental operator. However, if a command name was an acronym which the user was recalling letter by letter, then we would place one **M** operator per letter.

The physical operator times all depend on the skills of the user. Also, the mental operator depends on the level of chunking, and hence the expertise of the user. You must therefore decide before using KLM predictions just what sort of user you are thinking about. You cannot even work out the operators and then fill in the times later, as different users may choose different methods and have different placings of **M** operators due to chunking. This sounds rather onerous, but the predictions made by KLM are only meant to be an approximation, and thus reasonable guesses about levels of expertise are enough.

Individual predictions may be interesting, but the power of KLM lies in comparison. Given several systems, we can work out the methods to perform key tasks, and then use KLM to tell us which system is fastest. This is considerably cheaper than conducting lengthy experiments (levels of individual variation would demand vast numbers of trials – see Chapter 9). Furthermore, the systems need not even exist. From a description of a proposed system, we can predict the times taken for tasks. As well as comparing systems, we can compare methods within a system. This can be useful in preparing training materials, as we can choose to teach the faster methods.

Using the keystroke-level model

As an example, we compare the two methods for iconizing a window given in Section 12.2.1. One used the 'L7' function key, and the other the 'CLOSE' option from the window's pop-up menu. The latter is obtained by moving to the window's title bar, depressing the left mouse button, dragging the mouse down the pop-up menu to the 'CLOSE' option, and then releasing the mouse button. We assume that the user's hand is on the mouse to begin with, and hence only the `L7-METHOD` will require a homing operator. The operators for the two methods are as follows:

`L7-METHOD`	**H**[to keyboard] **MK**[L7 function key]
`CLOSE-METHOD`	**P**[to menu bar] **B**[LEFT down] **MP**[to option] **B**[LEFT up]

The total times are thus

$$\begin{aligned}\texttt{L7-METHOD} &= 0.4 + 1.35 + 0.28 \\ &= 2.03 \text{ seconds}\end{aligned}$$

$$\begin{aligned}\texttt{CLOSE-METHOD} &= 1.1 + 0.1 + 1.35 + 1.1 + 0.1 \\ &= 3.75 \text{ seconds}\end{aligned}$$

The first calculation is quite straightforward, but the second needs a little unpacking. The button presses are separate down and then up actions and thus each is timed at only 0.1 of a second, rather than 0.2 for a click, or 0.28 for typing. We have also used the simplified average of 1.1 seconds for the pointing task. From these predictions, we can see that the `L7-METHOD` is far faster. In Section 12.2.1, Sam's selection rule was to use the `L7-METHOD` when playing blocks. To do so, he can go on playing the game using the mouse in his right hand whilst moving his left hand over the key. Thus the real time for Sam, from when he takes his attention from the game to when the command is given, is less: 2.03 seconds minus the homing time, that is 1.63 seconds. Given the method is so fast, why does Sam not use it all the time?

Perhaps the average estimates for pointing times have biased our estimate. We can be a little more precise about the `CLOSE-METHOD` timing if we use Fitts' law instead of the average 1.1 seconds. The mouse will typically be in the middle of a 25 line high window. The title bar is 1.25 lines high. Thus the distance to target ratio for the first pointing task is 10:1. The 'CLOSE' option is four items down on the pop-up menu; hence the ratio for the second pointing task is 4:1. Thus we can calculate the pointing times:

$$\begin{aligned}\textbf{P}[\text{to menu bar}] &= 0.1 \log_2(10.5) = 0.339 \\ \textbf{P}[\text{to option}] &= 0.1 \log_2(4.5) = 0.217\end{aligned}$$

With these revised timings, KLM predicts the `CLOSE-METHOD` will take 2.1 seconds. So Sam's selection rule is not quite as bad as it initially seemed!

Worked exercise *Do a keystroke-level analysis for opening up an application in a visual desktop interface using a mouse as the pointing device, comparing at least two different methods for performing the task. Repeat the exercise using a trackball. Consider how the analysis would differ for various positions of the trackball relative to the keyboard and for other pointing devices.*

Answer We provide a keystroke-level analysis for three different methods for launching an application on a visual desktop. These methods are analyzed for a conventional one-button mouse, a trackball mounted away from the keyboard and one mounted close to the keyboard. The main distinction between the two trackballs is that the second one does not require an explicit repositioning of the hands, that is there is no time required for homing the hands between the pointing device and the keyboard.

Method 1 Double clicking on application icon

Steps	Operator	Mouse	$Trackball_1$	$Trackball_2$
1. Move hand to mouse	**H**[mouse]	0.400	0.400	0.000
2. Mouse to icon	**P**[to icon]	0.664	1.113	1.113
3. Double click	**2B**[click]	0.400	0.400	0.400
4. Return to keyboard	**H**[kbd]	0.400	0.400	0.000
Total times		1.864	2.313	1.513

Method 2 Using an accelerator key

Steps	Operator	Mouse	$Trackball_1$	$Trackball_2$
1. Move hand to mouse	**H**[mouse]	0.400	0.400	0.000
2. Mouse to icon	**P**[to icon]	0.664	1.113	1.113
3. Click to select	**B**[click]	0.200	0.200	0.200
4. Pause	**M**	1.350	1.350	1.350
5. Return to keyboard	**H**[kbd]	0.400	0.400	0.000
6. Press accelerator	**K**	0.200	0.200	0.200
Total times		3.214	3.663	2.763

Method 3 Using a menu

Steps	Operator	Mouse	$Trackball_1$	$Trackball_2$
1. Move hand to mouse	**H**[mouse]	0.400	0.400	0.000
2. Mouse to icon	**P**[to icon]	0.664	1.113	1.113
3. Click to select	**B**[click]	0.200	0.200	0.200
4. Pause	**M**	1.350	1.350	1.350
5. Mouse to file menu	**P**	0.664	1.113	1.113
6. Pop-up menu	**B**[down]	0.100	0.100	0.100
7. Drag to open	**P**[drag]	0.713	1.248	1.248
8. Release mouse	**B**[up]	0.100	0.100	0.100
9. Return to keyboard	**H**[kbd]	0.400	0.400	0.000
Total times		4.591	6.024	5.224

Card, Moran and Newell empirically validated KLM against a range of systems, both keyboard and mouse based, and a wide selection of tasks. The predictions were found to be remarkably accurate (an error of about 20%). KLM is thus one of the few models capable of giving accurate quantitative predictions about performance. However, the range of applications is correspondingly small. It tells us a lot about the micro-interaction, but not about the larger-scale dialog.

We have seen that you have to be quite careful, as the approximations you make can radically change the results – KLM is a guide, not an oracle. We should also add a word of caution about the assumption that fastest is best. There are certainly situations where this is so, for example highly repetitive tasks such as telephony or data entry. However, even expert users will often not use the fastest method. For example, the expert may have a set of general-purpose, non-optimal methods, rather than a few task-specific methods.

12.5.2 Three-state model

In Chapter 2, we saw that a range of pointing devices exists in addition to the mouse. Often these devices are considered logically equivalent, if the same inputs are available *to the application*. That is, so long as you can select a point on the screen, they are all the same. However, these different devices – mouse, trackball, light pen – feel very different. Although the devices are similar from the application's viewpoint, they have very different sensory–motor characteristics.

Buxton has developed a simple model of input devices [53], the *three-state model*, which captures some of these crucial distinctions. He begins by looking at a mouse. If you move it with no buttons pushed, it normally moves the mouse cursor about. This tracking behavior is termed state 1. Depressing a button over an icon and then moving the mouse will often result in an object being dragged about. This he calls state 2 (see Figure 12.1).

If instead we consider a light pen with a button, it behaves just like a mouse when it is touching the screen. When its button is not depressed, it is in state 1, and when its button is down, state 2. However, the light pen has a third state, when the light pen is not touching the screen. In this state the system cannot track the light pen's position. This is called state 0 (see Figure 12.2).

A touchscreen is like the light pen with no button. While the user is not touching the screen, the system cannot track the finger – that is, state 0 again. When the user touches the screen, the system can begin to track – state 1. So a touchscreen is a state 0–1 device whereas a mouse is a state 1–2 device. As there is no difference between a state 0–2 and a state 0–1 device, there are only the three possibilities we have seen.

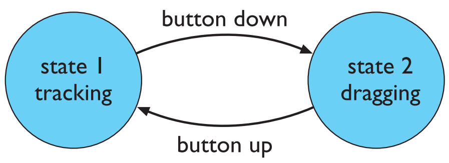


Figure 12.1 Mouse transitions: states 1 and 2

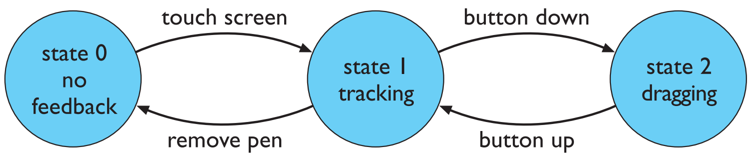


Figure 12.2 Light pen transitions: three states

The only additional complexity is if the device has several buttons, in which case we would have one state for each button: 2_{left}, 2_{middle}, 2_{right}.

One use of this classification is to look at different pointing tasks, such as icon selection or line drawing, and see what state 0–1–2 behavior they require. We can then see whether a particular device can support the required task. If we have to use an inadequate device, it is possible to use keyboard keys to add device states. For example, with a touchscreen, we may nominate the escape key to be the 'virtual' mouse button whilst the user's finger is on the screen. Although the mixing of keyboard and mouse keys is normally a bad habit, it is obviously necessary on occasions.

At first, the model appears to characterize the states of the device by the inputs available to the system. So, from this perspective, state 0 is clearly different from states 1 and 2. However, if we look at the state 1–2 transaction, we see that it is symmetric with respect to the two states. In principle, there is no reason why a program should not decide to do simple mouse tracking whilst in state 2 and drag things about in state 1. That is, there is no reason until you want to type something! The way we can tell state 1 from state 2 is by the activity of the *user*. State 2 requires a button to be pressed, whereas state 1 is one of relative relaxation (whilst still requiring hand–eye coordination for mouse movement). There is a similar difference in tension between state 0 and state 1.

It is well known that Fitts' law has different timing constants for different devices. Recall that Fitts' law says that the time taken to move to a target of size S at a distance D is:

$$a + b \log_2(D/S + 1)$$

The constants a and b depend on the particular pointing device used and the skill of the user with that device. However, the insight given by the three-state model is that these constants also depend on the device state. In addition to the timing, the final accuracy may be affected.

These observations are fairly obvious for state 0–1 devices. With a touchscreen, or light pen, a cursor will often appear under the finger or pen when it comes in contact with the screen. The accuracy with which you can move the cursor around will be far greater than the accuracy with which you can point in the first place. Also it is reasonable to expect that the Fitts' law constant will be different, although not so obvious which will be faster.

There is a similar difference between states 1 and 2. Because the user is holding a button down, the hand is in a state of tension and thus pointing accuracy and speed may be different. Experiments to calculate Fitts' law constants in states 1 and 2 have

Table 12.2 Fitts' law coefficients (after MacKenzie, Sellen and Buxton [221], © 1991 ACM, Inc. Reprinted by permission)

	Device	*a* (ms)	*b* (ms/bit)
Pointing (state 1)			
	Mouse	−107	223
	Trackball	75	300
Dragging (state 2)			
	Mouse	135	249
	Trackball	−349	688

shown that these differences do exist [221]. Table 12.2 shows the results obtained for a mouse and trackball.

We can recalculate the KLM prediction for the `CLOSE-METHOD` using these data. Recall that the method had two pointing operators, one to point to the window's title bar (with a distance to target size ratio of 10:1), the second to drag the selection down to 'CLOSE' on the pop-up menu (4:1). Thus the first pointing operator is state 1 and the second is state 2. The times are thus

Mouse

$$\begin{aligned} \text{P[to menu bar]} &= -107 + 223 \log_2(11) = 664 \text{ ms} \\ \text{P[to option]} &= 135 + 249 \log_2(5) = 713 \text{ ms} \end{aligned}$$

Trackball

$$\begin{aligned} \text{P[to menu bar]} &= 75 + 300 \log_2(11) = 1113 \text{ ms} \\ \text{P[to option]} &= -349 + 688 \log_2(5) = 1248 \text{ ms} \end{aligned}$$

giving a further revised time for the `CLOSE-METHOD` of 2.93 seconds using a mouse and 3.91 seconds using a trackball.

12.6 COGNITIVE ARCHITECTURES

The formalisms we have seen so far have some implicit or explicit model of how the user performs the cognitive processing involved in carrying out a task. For instance, the concept of taking a problem and solving it by divide and conquer using subgoals is central to GOMS. CCT assumes the distinction between long- and short-term memory, with production rules being stored in long-term memory and 'matched' against the contents of short-term (or working) memory to determine which 'fire'. The values for various motor and mental operators in KLM were based on the Model Human Processor (MHP) architecture of Card, Moran and Newell [56]. Another common assumption, which we have not discussed in this chapter, is the distinction between linguistic levels – semantic, syntactic and lexical – as an architectural model of the user's understanding.

In Chapter 1, we discussed some of these architectural descriptions of the user as an information-processing machine. Our emphasis in this section will be to describe a couple more architectural models that are quite distinct from those described in Chapter 1 and assumed in the earlier parts of this chapter. Here we will see that the architectural assumptions are central to the description of the cognitive modeling that these approaches offer.

There are interesting differences of emphasis between these architectural models and the previous models. The hierarchical and linguistic notations tend to assume perfect dialog on the user's part. They may measure the complexity of that perfect dialog, but tend not to consider diversions from the optimal command sequences. However, for the architectural models in this section the prediction and understanding of error is central to their analyses.

12.6.1 The problem space model

Rational behavior is characterized as behavior that is intended to achieve a specific goal. This element of rationality is often used to distinguish between intelligent and machine-like behavior. In the field of artificial intelligence (AI), a system exhibiting rational behavior is referred to as a *knowledge-level* system. A knowledge-level system contains an *agent* behaving in an environment. The agent has knowledge about itself and its environment, including its own goals. It can perform certain actions and sense information about its changing environment. As the agent behaves in its environment, it changes the environment and its own knowledge. We can view the overall behavior of the knowledge-level system as a sequence of environment and agent states as they progress in time. The goal of the agent is characterized as a preference over all possible sequences of agent/environment states.

Contrast this rational behavior with another general computational model for a machine, which is not rational. In computer science, it is common to describe a problem as the search through a set of possible states, from some initial state to a desired state. The search proceeds by moving from one state to another possible state by means of operations or actions, the ultimate goal of which is to arrive at one of the desired states. This very general model of computation is used in the ordinary task of the programmer. Once she has identified a problem and a means of arriving at the solution to the problem (the algorithm), the programmer then represents the problem and algorithm in a programming language, which can be executed on a machine to reach the desired state. The architecture of the machine only allows the definition of the search or *problem space* and the actions that can occur to traverse that space. Termination is also assumed to happen once the desired state is reached. Notice that the machine does not have the ability to formulate the problem space and its solution, mainly because it has no idea of the goal. It is the job of the programmer to understand the goal and so define the machine to achieve it.

We can adapt the state-based computational model of a machine in order to realize the architecture of a knowledge-level system. The new computational model is the *problem space model*, based on the problem-solving work of Newell and Simon at Carnegie–Mellon University (see Chapter 1). A problem space consists of a set of states and a set of operations that can be performed on the states. Behavior in a problem space is a two-step process. First, the current operator is chosen based on the current state and then it is applied to the current state to achieve the new state. The problem space must represent rational behavior, and so it must characterize the goal of the agent. A problem space represents a goal by defining the desired states as a subset of all possible states. Once the initial state is set, the task within the problem space is to find a sequence of operations that form a path within the state space from the initial state to one of the desired states, whereupon successful termination occurs.

From the above description, we can highlight four different activities that occur within a problem space: goal formulation, operation selection, operation application and goal completion. The relationship between these problem space processes and knowledge-level activity is key. Perception that occurs at the knowledge level is performed by the goal formulation process, which creates the initial state based on observations of the external environment. Actions at the knowledge level are operations in the problem space which are selected and applied. The real knowledge about the agent and its environment and goals is derived from the state/operator information in the problem space. Because of the goal formulation process, the set of desired states indicates the knowledge-level goal within the problem space. The operation selection process selects the appropriate operation at a given point in time because it is deemed the most likely to transform the state in the problem space to one of the desired states; hence rational behavior is implied.

The cycle of activity within the problem space is as follows. Some change in the external environment, which is relevant to the goal of the agent, is sensed by the goal formulation process, which in turn defines the set of desired states for the agent and its initial state for the following task. The operation selection process suggests an operation that can act on that state and transform it ‘closer’ to a desired state. The operation application process executes the operation, changing the current state and surrounding environment. If the new state is a desired state, then the goal has been achieved and the goal completion process reverts the agent to inactive.

The real power of the problem space architecture is in recursion. The activity of any of the processes occurs only when the knowledge it needs to complete its chore is immediately available. For example, to decide which operation is most likely to lead to a desired state, the problem space will need to know things about its current state and that of the environment. If that information is not immediately available, then activity cannot continue. In that case, another problem space is invoked with the goal of finding out the information that was needed by the parent problem space. In this way, we can see the evolution of problems spaces as a stack-like structure, new spaces being invoked and placed on the problem space stack only to be popped off the stack once they achieve their goal.

Though the problem space model described briefly above is not directly implementable, it is the basis for at least one executable cognitive architecture, called Soar.

We shall not discuss the details of Soar's implementation here; instead we refer you to Laird, Newell and Rosenbloom [205]. An interesting application of the Soar implementation of problem spaces has been done by Young and colleagues on *programmable user models* (or *PUMs*) [386]. Given a designer's description of an intended procedure or task that is to be carried out with an interactive system, an analysis of that procedure produces the knowledge that would be necessary and available for any user attempting the task. That knowledge is encoded in the problem space architecture of Soar, producing a 'programmed' user model (the PUM) to accomplish the goal of performing the task. By executing the PUM, the stacking and unstacking of problem spaces needed to accomplish the goal can be analyzed to measure the cognitive load of the intended procedure. More importantly, if the PUM cannot achieve the goal because it cannot find some knowledge necessary to complete the task, this indicates to the designer that there is a problem with the intended design. In this way, erroneous behavior can be predicted before implementation.

12.6.2 Interacting cognitive subsystems

Barnard has proposed a very different cognitive architecture, called *interacting cognitive subsystems* (*ICS*) [24, 25, 27]. ICS provides a model of perception, cognition and action, but unlike other cognitive architectures, it is not intended to produce a description of the user in terms of sequences of actions that he performs. ICS provides a more holistic view of the user as an information-processing machine. The emphasis is on determining how easy particular procedures of action sequences become as they are made more automatic within the user.

ICS attempts to incorporate two separate psychological traditions within one cognitive architecture. On the one hand is the architectural and general-purpose information-processing approach of short-term memory research. On the other hand is the computational and representational approach characteristic of psycholinguistic research and AI problem-solving literature.

The architecture of ICS is built up by the coordinated activity of nine smaller subsystems: five peripheral subsystems are in contact with the physical world and four are central, dealing with mental processes. Each subsystem has the same generic structure. A subsystem is described in terms of its typed inputs and outputs along with a memory store for holding typed information. It has transformation functions for processing the input and producing the output and permanently stored information. Each of the nine subsystems is specialized for handling some aspect of external or internal processing. For example, one peripheral subsystem is the visual system for describing what is seen in the world. An example of a central subsystem is one for the processing of propositional information, capturing the attributes and identities of entities and their relationships with each other (a particular example is that propositional information represents '"knowing" that a particular word has four syllables, begins with "P" and refers to an area in central London' [27]).

ICS is another example of a general cognitive architecture that can be applied to interactive design. One of the features of ICS is its ability to explain how a user proceduralizes action. Remember in the discussion of CCT we distinguished between novice and expert use of an interactive system. Experts can perform complicated sequences of actions as if without a thought, whereas a novice user must contemplate each and every move (if you do not believe this distinction is accurate, observe users at an ATM and see if you can tell the expert from the novice). The expert recognizes the task situation and recalls a ‘canned’ procedure of actions which, from experience, results in the desired goal being achieved. They do not have to think beyond the recognition of the task and consequent invocation of the correct procedure. Such proceduralized behavior is much less prone to error. A good designer will aid the user in proceduralizing his interaction with the system and will attempt to design an interface that suggests to the user a task for which he already has a proceduralized response. It is for this reason that ICS has been suggested as a design tool that can act as an expert system to advise a designer in developing an interface.

12.7 SUMMARY

In this chapter, we have discussed a wide selection of cognitive models of the users of interactive systems. Cognitive models attempt to represent the users as they interact with a system, modeling aspects of their understanding, knowledge, intentions or processing. We divided cognitive models into three categories. The first described the hierarchical structuring of the user’s task and goal structures. The GOMS model and CCT were examples of cognitive models in this category. The second category was concerned with linguistic and grammatical models, which emphasized the user’s understanding of the user–system dialog. BNF and TAG were described as examples in this category. Most of these cognitive models have focussed on the execution activity of the user, neglecting his perceptive ability and how that might affect less planned and natural interaction strategies. The third category of cognitive models was based on the more solid understanding of the human motor system, applicable in situations where the user does no planning of behavior and executes actions automatically. KLM was used to provide rough measures of user performance in terms of execution times for basic sequences of actions. Buxton’s three-state model for pointing devices allowed for a finer distinction between execution times than with KLM. We concluded this chapter with a discussion of cognitive architectures, the assumptions of which form the foundation for any cognitive models. In addition to the basic architectural distinction between long- and short-term memory, we discussed two other cognitive architectures – the problem space model and ICS – which apply different assumptions to the analysis of interactive systems.

EXERCISES

12.1 Recall the CCT description of the rule `INSERT-SPACE-2` discussed in Section 12.2.2:

```
(INSERT-SPACE-2
IF (AND (TEST-GOAL insert space)
                (TEST-CURSOR %LINE %COL) )
THEN    (       (DO-KEYSTROKE 'I')
                (DO-KEYSTROKE SPACE)
                (DO-KEYSTROKE ESC)
                (DELETE-GOAL insert space) ))
```

As we discussed, this is already proceduralized; that is, the rule is an expert rule. Write new 'novice' rules where the three keystrokes are not proceduralized. That is, you should have separate rules for each keystroke and suitable goals (such as `GET-INTO-INSERT-MODE`) to fire them.

12.2 One of the assumptions underlying the programmable user model approach is that it is possible to provide an algorithm to describe the user's behavior in interacting with a system. Taking this position to the extreme, choose some common task with a familiar interactive system (for example, creating a column of numbers in a spreadsheet and calculating their sum, or any other task you can think of) and describe the algorithm needed by the user to accomplish this task. Write the description in pseudocode. Does this exercise suggest any improvements in the system?

RECOMMENDED READING

S. K. Card, T. P. Moran and A. Newell, *The Psychology of Human Computer Interaction*, Lawrence Erlbaum, 1983.
A classic text in this field of cognitive models, in which the basic architectural assumptions of the Model Human Processor architecture are explained as well as the GOMS model and KLM.

S. Bovair, D. E. Kieras and P. G. Polson, The acquisition and performance of text-editing skill: a cognitive complexity analysis, *Human–Computer Interaction*, Vol. 5, No. 1, pp. 1–48, 1990.
This article provides a definitive description of CCT by means of an extended example. The authors also provide the definition of various style rules for writing CCT descriptions to distinguish, for example, between novice and expert users.

F. Schiele and T. Green, HCI formalisms and cognitive psychology: the case of task–action grammars. In M. D. Harrison and H. W. Thimbleby, editors, *Formal Methods in Human–Computer Interaction*, Chapter 2, Cambridge University Press, 1990.

A good description of TAG with several extended examples based on the Macintosh interface. The authors provide a good comparative analysis of TAG versus other cognitive modeling techniques.

A. Newell, G. Yost, J. E. Laird, P. S. Rosenbloom and E. Altmann, Formulating the problem-space computational model. In R. F. Rashid, editor, *CMU Computer Science: A 25th Anniversary Commemorative*, Chapter 11, ACM Press, 1991.
The description of the problem space cognitive architecture was informed by this article, which also contains references to essential work on the Soar platform.

J. Carroll, editor, *HCI Models, Theories, and Frameworks: Toward an Interdisciplinary Science*, Morgan Kaufmann, 2003.
See chapters by Scott MacKenzie on motor behavior models (Fitts' law and beyond), Bonnie John on information processing and skilled behavior (GOMS and related models) and Stephen Payne on cognitive and mental models.

13 Socio-organizational issues and stakeholder requirements

OVERVIEW

- There are several organizational issues that affect the acceptance of technology by users and that must therefore be considered in system design:
  - systems may not take into account conflict and power relationships
  - those who benefit may not do the work
  - not everyone may use systems.
- In addition to generic issues, designers must identify specific stakeholder requirements within their organizational context.
- Socio-technical models capture both human and technical requirements.
- Soft systems methodology takes a broader view of human and organizational issues.
- Participatory design includes the user directly in the design process.
- Ethnographic methods study users in context, attempting to take an unbiased perspective.

13.1 INTRODUCTION

As we saw in Chapter 3, technology does not exist in a vacuum. It is used within a specific context, and is influenced by many factors within that context. The different people affected by the introduction of a system are known as stakeholders and their needs can be both complex and conflicting. In addition, we need to understand how the introduction of the system might actually change the organizational and work practices that are currently in place and what the impact of this might be.

In this chapter we look in more detail at the socio-organizational context of use and discuss some of the generic issues that can affect the acceptance of technology in organizations. We then look at a number of approaches to modeling the socio-organizational context and the requirements of the stakeholders within it.

Requirements capture is an important part of all software engineering methodologies but often this activity focusses primarily on the functional requirements of the system – what the system must be able to do – with less emphasis on non-functional human issues such as usability and acceptability. Even where such matters are considered, they may reflect only the management's view of the user's needs rather than gathering information from the users themselves. Stakeholder requirements modeling redresses this balance by identifying the needs of all stakeholders, including the user and anyone else affected by the system, within the context in which it will be used.

We will begin this chapter by discussing some of the organizational issues that arise when new technological solutions are introduced. We then outline a number of models and methods that can be used to capture this broader view of stakeholder requirements, including socio-technical models, soft systems methodology, participatory design and the ethnographic approach.

13.2 ORGANIZATIONAL ISSUES

In this section, we shall look at some of the organizational issues that affect the acceptance and relevance of information and communication systems. These factors often sit 'outside' the system as such, and may involve individuals who never use it. Yet it is often these factors more than any other that determine the success or failure of computer systems. Many systems supporting work in organizations are supporting groups of workers, but this may be through specialist groupware systems (see Chapter 19) or through shared data or processes.

13.2.1 Cooperation or conflict?

The term 'computer-supported *cooperative* work' (CSCW) seems to assume that groups will be acting in a cooperative manner. This is obviously true to some extent;

even opposing football teams cooperate to the extent that they keep (largely) within the rules of the game, but their cooperation only goes so far. People in organizations and groups have conflicting goals, and systems that ignore this are likely to fail spectacularly.

Imagine that an organization is already highly computerized, the different departments all have their own systems and the board decides that an integrated information system is needed. The production manager can now look directly at stocks when planning the week's work, and the marketing department can consult the sales department's contact list to send out marketing questionnaires. All is rosy and the company will clearly run more efficiently – or will it?

The storekeeper always used to understate stock levels slightly in order to keep an emergency supply, or sometimes inflate the quoted levels when a delivery was due from a reliable supplier. Also, requests for stock information allowed the storekeeper to keep track of future demands and hence plan future orders. The storekeeper has now lost a sense of control and important sources of information. Members of the sales department are also unhappy: their contacts are their livelihood. The last thing they want is someone from marketing blundering in and spoiling a relationship with a customer built up over many years. Some of these people may resort to subverting the system, keeping 'sanitized' information online, but the real information in personal files. The system gradually loses respect as the data it holds is incorrect, morale in the organization suffers and productivity drops. The board gets worried and meets to consider upgrading the computer system!

Before installing a new computer system, whether explicitly 'cooperative' or not, one must identify the *stakeholders* who will be affected by it. These are not just the immediate users, but anyone whose jobs will be altered, who supplies or gains information from it, or whose power or influence within the organization will increase or decrease. It will frequently be the case that the formal 'client' who orders the system falls very low on the list of those affected. Be very wary of changes that take power, influence or control from some stakeholders without returning something tangible in their place.

13.2.2 Changing power structures

The identification of stakeholders will uncover information transfer and power relationships that cut across the organizational structure. Indeed, all organizations have these informal networks that support both social and functional contacts. However, the official lines of authority and information tend to flow up and down through line management. New communications media may challenge and disrupt these formal managerial structures.

The physical layout of an organization often reflects the formal hierarchy: each department is on a different floor, with sections working in the same area of an office. If someone from sales wants to talk to someone from marketing then one of them must walk to the other's office. Their respective supervisors can monitor the contact. Furthermore, the physical proximity of colleagues can foster a sense of departmental loyalty. An email system has no such barriers; it is as easy to 'chat' to someone in another department as in your own. This challenges the mediating and controlling role of the line managers.

Furthermore, in face-to-face conversation, the manager can easily exert influence over a subordinate: both know their relative positions and this is reflected in the patterns of conversation and in other non-verbal cues. Email messages lose much of this sense of presence and it is more difficult for a manager to exercise authority. The 'levelling' effect even makes it possible for subordinates to direct messages 'diagonally' across the hierarchy, to their manager's peers, or, even worse, to their manager's manager!

Many organizations are moving toward flatter management structures anyway, so from a strategic viewpoint these effects may be acceptable. But can the organization cope with a disaffected junior management during the transition? For other organizations the effects may be less welcome and the system dropped or heavily regulated. In one case, an email system was introduced and was agreed to be functioning well, but the management, feeling a loss of control and suspicion over their subordinates' communications, introduced logging so that all email messages could be monitored. The system quickly fell into disuse. Logging of email is becoming more widespread with employers using it to identify cases of system 'abuse' by employees. But such activity must be handled carefully: it is as likely to backfire on the management by reducing the productive use of email as it is to have the desired effect.

Technology can be an important vector of social change, but if violent reaction is to be avoided, the impact of the technology must be assessed before it is introduced. In the short term, solutions must be carefully matched to the existing social and organizational structures.

13.2.3 The invisible worker

The ability to work and collaborate at a distance can allow functional groups to be distributed over different sites. This can take the form of cross-functional neighbourhood centers, where workers from different departments do their jobs in electronic contact with their functional colleagues. Alternatively, distributed groupware can allow the true home-based teleworker to operate on similar terms to an office-based equivalent. The ecological and economic advantages of such working practices are now becoming well established, and it seems that communications and CSCW technology can overcome many of the traditional barriers.

In fact, a closer examination reveals that the barriers to such working are not technological but managerial. First of all, management style may make remote working all but impossible. If the approach in an organization is 'management by presence', that is you know someone is working because they are in the office, then there is no way a remote worker is going to be trusted. If, on the other hand, the style is 'management by objectives', that is you know your subordinates are working because they are doing their jobs and producing results, then remote working is not so problematical. Even where remote working is accepted, the lack of physical presence can be a problem. When the time comes for promotion, the present employee may seem more worthy than the distant one – not because of any objective criteria, but because presence increases perceived worth.

We can see that, again, social and managerial relationships completely dominate technological considerations. Many video-based groupware systems are intended to create a sense of engagement, of active participation and social presence. Whether this will be sufficient to overcome ingrained attitudes remains to be seen.

13.2.4 Who benefits?

One frequent reason for the failure of information systems is that the people who get the benefits from the system are not the same as those who do the work. One example, which we discuss in more detail in Chapter 19, is structured message systems such as Lens. In these systems the sender has to do work in putting information into fields appropriately, but it is the recipient who benefits. Another example is shared calendars. The beneficiary of the system is a manager who uses the system to arrange meeting times, but whose personal secretary does the work of keeping the calendar up to date. Subordinates are less likely to have secretarial support, yet must keep up the calendar with little perceived benefit. Of course, chaos results when a meeting is automatically arranged and the subordinates may have to rearrange commitments that have not been recorded on the system. The manager may force use by edict or the system may simply fall into disuse. Many such groupware systems are introduced on a 'see if it works' basis, and so the latter option is more likely.

The lesson is that information systems should aim for some level of *symmetry*. If you have to do work for the system, you should obtain some benefit from it. For the shared calendar, this might involve improving the personal user interface, so that there are definite advantages in using the online system to plan your time rather than using paper (it could even print out Filofax pages). In addition, if people use electronic organizers one could consider integrating these into the system.

13.2.5 Free rider problem

Even where there is no bias toward any particular people, a system may still not function symmetrically, which may be a problem, particularly with shared communication systems. One issue is the *free rider problem*. Take an electronic conferencing system. If there is plenty of discussion of relevant topics then there are obvious advantages to subscribing and reading the contributions. However, when considering writing a contribution, the effort of doing so may outweigh any benefits. The total benefit of the system for each user outweighs the costs, but for any particular decision the balance is overturned.

To see this situation in a different context imagine an eccentric philanthropist who has gathered three strangers into a room. They are invited to throw money into a pot in the center. When they have done so, the philanthropist will double the money in the pot and then divide it up between them and send them on their way. Each stranger reasons 'If I put in three pennies, then our benefactor will double this to six. These will be distributed between three of us, so I will have only two returned to me.' Clearly, unless the strangers can come to some understanding none of them will put any money in the pot, and none will benefit.

A few free riders in a conference system are often not a problem, as the danger is more likely from too much activity. In addition, in electronic conferences the patterns of activity and silence may reflect other factors such as expertise. However, it is easy for the number of free riders gradually to increase and the system slide into disuse. It is hard to enforce equal use, except by restrictive schemes such as round-robin contributions (everyone contributes something however short). In the real world, such problems are often solved by social pressure, and the free rider reacts to the collective censure of the group. Increasing the *visibility* of participants' contributions might also help these social mechanisms. For example, one could display an activity meter showing the number of contributions from each subscriber. Of course, people may then choose not to subscribe in the first place!

13.2.6 Critical mass

Another issue related to the free rider problem is the need to develop a *critical mass*. When telephones were only in public places, their use as a form of pervasive interpersonal communication was limited. However, once a large number of people have telephones in their homes it becomes worthwhile paying to have a telephone installed. In cost/benefit terms, the early subscribers probably have a smaller benefit than the cost. Only when the number of subscribers increases beyond the critical mass does the benefit *for all* dominate the cost (see Figure 13.1). The situation for conferencing systems and email is, of course, very similar.

We can learn something from the lessons of the telephone system and other successful technologies (but remember, telephones took the best part of 100 years to become pervasive in affluent countries). The telephone was useful for subgroups before it became beneficial for all. Even when only a small proportion of the

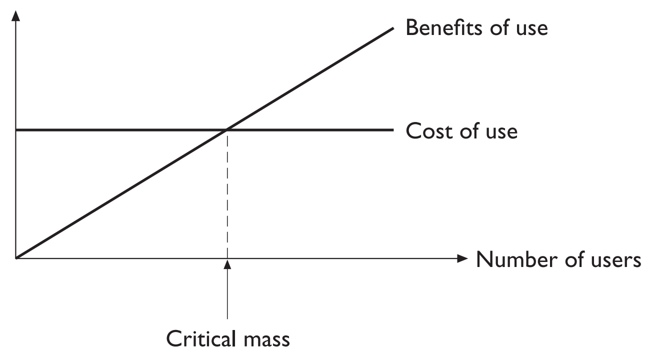


Figure 13.1 Cost/benefit of system use

population had personal telephones, they still formed a significant proportion of their social group, so these cliques of use could grow gradually over time.

The same is true of email. Even a group of two or three people in an organization can use email effectively between themselves. If an organization consists of widening circles of highly connected subgroups, then take-up can grow from the core to the wider group. Of course, the benefits increase as it becomes pervasive, but even a tiny user group ensures that the benefits outweigh the costs. Clearly, we must design any new system so that it has benefits even when its user population is small.

13.2.7 Automating processes – workflow and BPR

The major task in many organizations is moving pieces of paper around. An order is received by phone and an order form filled in by the sales executive. The order form is passed to accounts who check the credit rating and if all is okay it is passed on to stores who check availability and collect the order together at the picking line. When the order is dispatched, a delivery note is packed with the order and a copy is returned to accounts, who send an invoice to the customer.

Organizations have many such processes, and *workflow* systems aim to automate much of the process using electronic forms, which are forwarded to the relevant person based on pre-coded rules. Some workflow systems are built using special-purpose groupware, often based on a notation for describing the desired workflow. Coordinator (see Section 14.3.6) is an example of a workflow system where the rules are heavily influenced by speech act theory. In addition, workflow systems may be implemented using more general structured message systems, including Lotus Notes (see the Design Focus in this section).

The rigid form of a typical workflow system is an example of *global structuring* (see Chapter 19 for more details). The danger with any form of global structuring is that it may conflict with or inhibit more informal and less structured patterns of activity which also contribute to the organization's free running.

A more radical approach to organizational processes is found in *business process re-engineering* (*BPR*). Traditionally, organizations have been structured around functions: sales, accounts, stores, manufacturing. However, the purpose of an organization can be seen in terms of key business processes. The ordering/delivery process described above is a typical and important example. In BPR these processes are recorded and analyzed. Problems in the current process are noted and the whole process may be redesigned in order to make the path of the process more efficient. For example, instead of sending an order to the accounts department to approve, a list of customer credit limits could be given to the sales executives. They could then check the credit rating of the customer whilst on the phone and only forward the order to accounts if there are any unusual problems. Finally, and most radically, the whole structure of the organization may be modified to reflect and support the key processes more closely. Typically, this involves stripping layers of middle management. BPR as an issue engenders zealots and reactionaries in equal measure.

Design Focus

Implementing workflow in Lotus Notes

Lotus Notes can be used to implement workflow systems in a straightforward manner. The sales executive fills in an electronic form which is automatically emailed to the accounts department. When it is approved the order form is automatically emailed to stores, and so on.

However, interruptions due to email can be disruptive and reduce efficiency. An alternative is actively not to mail items for action to people, but instead to make them available within their electronic environment. This is like the difference between having a form for approval pushed in front of you to sign compared with having a pile of order forms sitting in your in-tray. In the latter case you can organize your own work, but cannot forget to process the forms as they sit as a permanent reminder – an *environmental cue*.

Within the electronic world of Notes such environmental cues can be implemented using its views mechanism. Each database can have associated several views, lists of items in the database. The definition of the view specifies which fields of each item should be listed and also which items should be included in the list, based on various criteria. The simplest filtering criterion is the type of the message, but more complex criteria can include the state of various fields.

In the ordering process example, there can be a single database. When an order is received, the sales executive creates a new order item and fills in the details of the customer and required products. However, the sales executive does not have write permission for the 'approved' field. The accounts department has a view of the same database. This view lists only those orders which have completed customer details, but a blank 'approved' field. Thus all orders received appear in this listing until they have the 'approved' field completed, at which point they disappear from the view (but not the database). Similarly, the stores department has a view whose rule only shows orders with a completed 'approved' field. Thus the views act as a sort of to-do list.

In the first scenario, the workflow is *automated* and control rests in the system. In the second scenario, the workflow is *supported* leaving control to the users.

13.2.8 Evaluating the benefits

We have seen several problems that can arise from the mismatch between information systems and organizational and social factors. Let us assume that we have a system in place – and it has not fallen apart at the seams. Everyone seems happy with it and there are no secret resentments. Now it is time to count the cost – it was an expensive system to buy and install, but was it worth it?

This is an almost impossible question to answer. The benefits from cooperative systems, especially organization-wide systems such as email or electronic conferencing, are in terms of job satisfaction or more fluid information flow. Some, such as the *video wall* (see Chapter 19), are expected primarily to help social contact within the organization. It may be possible to measure contentment and job satisfaction using attitude questionnaires (see Chapter 9), but any hard economic benefit will be so diffuse as to be unquantifiable.

However, a similar argument could be (and has been) framed for computer use in general. The benefits are difficult to quantify, but, over time, it has become clear that the competitive edge of information technology is necessary for survival in the modern world. Perhaps the same will be said of cooperative systems in a few years.

13.3 CAPTURING REQUIREMENTS

As we have already seen, problems can arise when a system is introduced without a full understanding of all the people who will be affected by it. But how can we better understand and support complex organizational structures, workgroups and potentially conflicting stakeholder needs? We begin by capturing and analyzing requirements, but we need to do this within the work context, taking account of the complex mix of concerns felt by different stakeholders and the structures and processes operating in the workgroups.

In this section we consider several approaches: socio-technical modeling, soft systems methodology, participatory design, ethnographic methods and contextual inquiry. All are aimed at understanding the reality of work contexts and the perspectives of different stakeholders. All recognize that technology can be successfully deployed only if it is done with an understanding of the context of use, but each takes a slightly different approach to the problem. Before we look in more detail at these approaches we need to clarify what we mean when we talk about ‘stakeholders’.

13.3.1 Who are the stakeholders?

Understanding stakeholders is key to many of the approaches to requirements capture, since in an organizational setting it is not simply the end-user who is affected by the introduction of new technology. Imagine that a new billing system is to be introduced by a local gas supplier. Who will be affected by this decision? Obviously, the people who are responsible for producing and sending out bills – they will be the ones using the system directly. But where do they get the information from to produce the bills? To whom do they send the bills? Who determines the level of charging and on what grounds? Who stands to make a profit from increased revenue? Who will suffer if customers choose to switch supplier due to the improved service? Meter readers, customers, managers, regulators, shareholders and competitors are all stakeholders in the system. We need approaches that will capture the complexity of their concerns, which may be in conflict with each other.

A stakeholder, therefore, can be defined as anyone who is affected by the success or failure of the system. It can be useful to distinguish different categories of stakeholder, and the following categorization from the CUSTOM approach (see [200]) is helpful for this:

Primary stakeholders are people who actually use the system – the end-users.

Secondary stakeholders are people who do not directly use the system, but receive output from it or provide input to it (for example, someone who receives a report produced by the system).

Tertiary stakeholders are people who do not fall into either of the first two categories but who are directly affected by the success or failure of the system (for example, a director whose profits increase or decrease depending on the success of the system).

Facilitating stakeholders are people who are involved with the design, development and maintenance of the system.

Classifying stakeholders – an airline booking system

An international airline is considering introducing a new booking system for use by associated travel agents to sell flights directly to the public. The stakeholders can be classified as follows:

- **Primary stakeholders:** travel agency staff, airline booking staff
- **Secondary stakeholders:** customers, airline management
- **Tertiary stakeholders:** competitors, civil aviation authorities, customers' traveling companions, airline shareholders
- **Facilitating stakeholders:** design team, IT department staff

The aim of the design team is to meet the needs of as many stakeholders as possible. However, the reality is that usually stakeholder needs are in conflict with each other. Sometimes this does not matter: a company is unlikely to be too concerned that its competitors' requirement to maintain advantage over it is under threat by the new system (though they need to be aware to monitor how effectively they are maintaining their advantage). However, sometimes it does matter. In the example above, the airline booking system must be usable by travel agency staff but must also fulfill the customer need to find an appropriate ticket at the right price. If it fails in this, the whole system will fail, as the customer will go elsewhere and business will be lost.

As a general rule, the priority of stakeholder needs diminishes as you go down the categories. So primary stakeholders usually take priority over the others. However, this is not always the case. Imagine designing the control panel of a hospital life support machine. The primary stakeholders will be medical staff monitoring a patient's condition. But who, in fact, has the greatest interest in this system working? Surely it is the patient, whose life is dependent on the system's success? In this case the tertiary stakeholder is of vital importance.

All of the approaches we are considering here are concerned with understanding stakeholders within their organizational context.

13.3.2 Socio-technical models

Early in the twentieth century, studies of work focussed on how humans needed to adapt to technical innovations. *Technological determinism*, the view that social change is primarily dictated by technology, with human and social factors being secondary concerns, was prevalent. The socio-technical systems view came about to counter this technology-centric position, by stressing that work systems were composed of both human and machine elements and that it was the interrelationship between these that should be central.

Socio-technical models for interactive systems are therefore concerned with technical, social, organizational and human aspects of design. They recognize the fact that technology is not developed in isolation but as part of a wider organizational environment. It is important to consider social and technical issues side by side so that human issues are not overruled by technical considerations.

The key focus of the socio-technical approach is to describe and document the impact of the introduction of a specific technology into an organization. Methods vary but most attempt to capture certain common elements:

- The problem being addressed: there is a need to understand why the technology is being proposed and what problem it is intended to solve.
- The stakeholders affected, including primary, secondary, tertiary and facilitating, together with their objectives, goals and tasks.
- The workgroups within the organization, both formal and informal.
- The changes or transformations that will be supported.
- The proposed technology and how it will work within the organization.
- External constraints and influences and performance measures.

Information is gathered using methods such as interviews, observation, focus groups and document analysis. The methods guide this information-gathering process and help the analyst to make sense of what is discovered. By attempting to understand these issues, socio-technical approaches aim to provide a detailed view of the role technology will play and the requirements of successful deployment.

We will compare two approaches to illustrate how this may work in practice.

CUSTOM methodology

CUSTOM is a socio-technical methodology designed to be practical to use in small organizations [200]. It is based on the User Skills and Task Match (USTM) approach, developed to allow design teams to understand and fully document user requirements [219]. CUSTOM focusses on establishing stakeholder requirements: all stakeholders are considered, not just the end-users.

It is applied at the initial stage of design when a *product opportunity* has been identified, so the emphasis is on capturing requirements. It is a forms-based methodology, providing a set of questions to apply at each of its stages.

There are six key stages to carry out in a CUSTOM analysis:

1. Describe the organizational context, including its primary goals, physical characteristics, political and economic background.
2. Identify and describe stakeholders. All stakeholders are named, categorized (as primary, secondary, tertiary or facilitating) and described with regard to personal issues, their role in the organization and their job. For example, CUSTOM addresses issues such as stakeholder motivation, disincentives, knowledge, skills, power and influence within the organization, daily tasks and so on.
3. Identify and describe work-groups. A work-group is any group of people who work together on a task, whether formally constituted or not. Again, work-groups are described in terms of their role within the organization and their characteristics.
4. Identify and describe task–object pairs. These are the tasks that must be performed, coupled with the objects that are used to perform them or to which they are applied.
5. Identify stakeholder needs. Stages 2–4 are described in terms of both the current system and the proposed system. Stakeholder needs are identified by considering the differences between the two. For example, if a stakeholder is identified as currently lacking a particular skill that is required in the proposed system then a need for training is identified.
6. Consolidate and check stakeholder requirements. Here the stakeholder needs list is checked against the criteria determined at earlier stages.

Stages 2 to 4 are described in terms of the current situation (before the new technology is introduced) and the proposed situation (after deployment). Stakeholders are asked to express their views not only of their current role and position but of their expectations in the light of the changes that will be made. In this way, stakeholder concerns and goals are elaborated. In addition, the impact of the technology on working practices is considered (Stage 3) and the transformations that will be supported by the system specified (Stage 4).

A shorter version of CUSTOM stakeholder analysis

CUSTOM questions investigate a range of stakeholder characteristics, such as the following:

- What does the stakeholder have to achieve and how is success measured?
- What are the stakeholder's sources of job satisfaction? What are the sources of dissatisfaction and stress?
- What knowledge and skills does the stakeholder have?
- What is the stakeholder's attitude toward work and computer technology?
- Are there any work-group attributes that will affect the acceptability of the product to the stakeholder?
- What are the characteristics of the stakeholder's task in terms of frequency, fragmentation and choice of actions?
- Does the stakeholder have to consider any particular issues relating to responsibility, security or privacy?
- What are the physical conditions in which the stakeholder is working?

The changes from the current position to the proposed position represent the issues that need to be addressed to ensure successful deployment, and these are made explicit during Stages 5 and 6.

CUSTOM provides a useful framework for considering stakeholder requirements and the use of forms and questions (a 'manual' for its use is available [200]) makes it relatively straightforward, if somewhat time consuming, to apply. For less complex situations, a shortened version of CUSTOM stakeholder analysis is available (see the boxed text above). This also provides a checklist for investigations for stages 2–4.

Open System Task Analysis (OSTA)

OSTA [116] is an alternative socio-technical approach, which attempts to describe what happens when a technical system is introduced into an organizational work environment. Like CUSTOM, OSTA specifies both social and technical aspects of the system. However, whereas in CUSTOM these aspects are framed in terms of stakeholder perspectives, in OSTA they are captured through a focus on tasks.

OSTA has eight main stages:

1. The primary task which the technology must support is identified in terms of users' goals.
2. Task inputs to the system are identified. These may have different sources and forms that may constrain the design.
3. The external environment into which the system will be introduced is described, including physical, economic and political aspects.
4. The transformation processes within the system are described in terms of actions performed on or with objects.
5. The social system is analyzed, considering existing work-groups and relationships within and external to the organization.
6. The technical system is described in terms of its configuration and integration with other systems.
7. Performance satisfaction criteria are established, indicating the social and technical requirements of the system.
8. The new technical system is specified.

OSTA uses notations familiar to designers, such as data flow diagrams and textual descriptions.

13.3.3 Soft systems methodology

The socio-technical models we have looked at focus on identifying requirements from both human and technical perspectives, but they assume a technological solution is being proposed. Soft systems methodology (SSM) arises from the same tradition but takes a view of the organization as a system of which technology and people are components. There is no assumption of a particular solution: the emphasis is rather on understanding the situation fully. SSM was developed by Checkland [66]

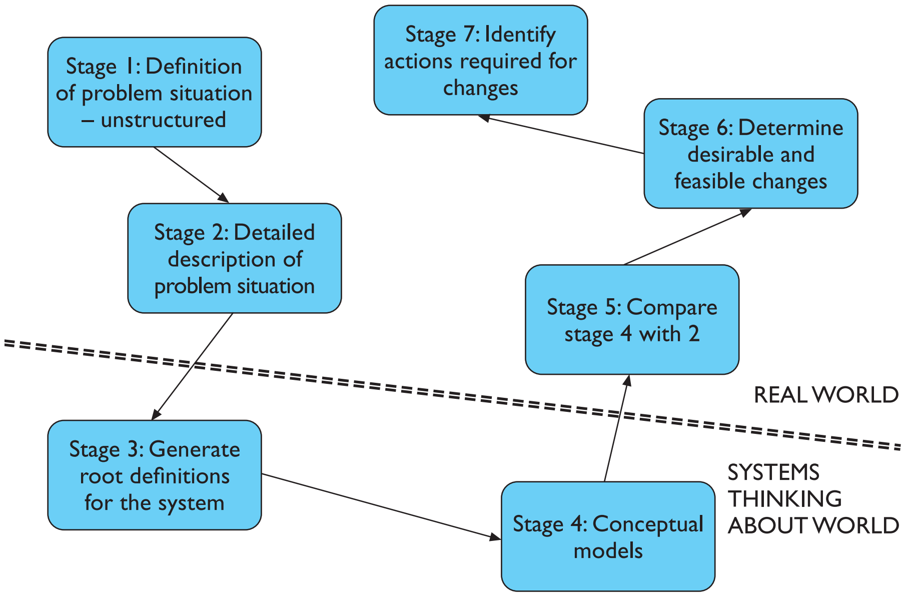

Stage 1: Definition of problem situation – unstructured
Stage 2: Detailed description of problem situation
Stage 3: Generate root definitions for the system
Stage 4: Conceptual models
Stage 5: Compare stage 4 with 2
Stage 6: Determine desirable and feasible changes
Stage 7: Identify actions required for changes
REAL WORLD
SYSTEMS THINKING ABOUT WORLD

Figure 13.2 The seven stages of soft systems methodology. (Adapted from Checkland [66], p.163)

to help designers reach an understanding of the context of technological developments and the influences and concerns that exist within the system under consideration. SSM has seven stages (see Figure 13.2). A distinction is made between the 'real-world' stages (1–2, 5–7) and the systems stages (3–4).

We will outline the stages here but will focus on those that help capture requirements. The first stage of SSM is the recognition of the problem and initiation of analysis. This is followed by a detailed description of the problem situation: developing a *rich picture*. This will include all the stakeholders, the tasks they carry out and the groups they work in, the organizational structure and its processes and the issues raised by each stakeholder. Any knowledge elicitation techniques can be used to gather the information to build the rich picture, including observation (and video and audio recording), structured and unstructured interviews and questionnaires, and workshops incorporating such activities as role play, simulations and critical incident analysis. In general, less structured approaches are used initially to avoid artificially constraining the description. The rich picture can be in any style – there are no right or wrong answers – but it should be clear and informative to the designer. Certain conventions are widely accepted, however. Speech balloons are used to represent stakeholder issues; crossed swords represent conflicts within the system; and the eye represents external influences or observers. Figure 13.3 shows an example of a rich picture with all of these elements.

Rich pictures are in themselves useful tools to aid understanding of a situation. The rich picture is informal and relatively intuitive. It captures succinctly the potentially conflicting interests of the various stakeholders and the other influences on a

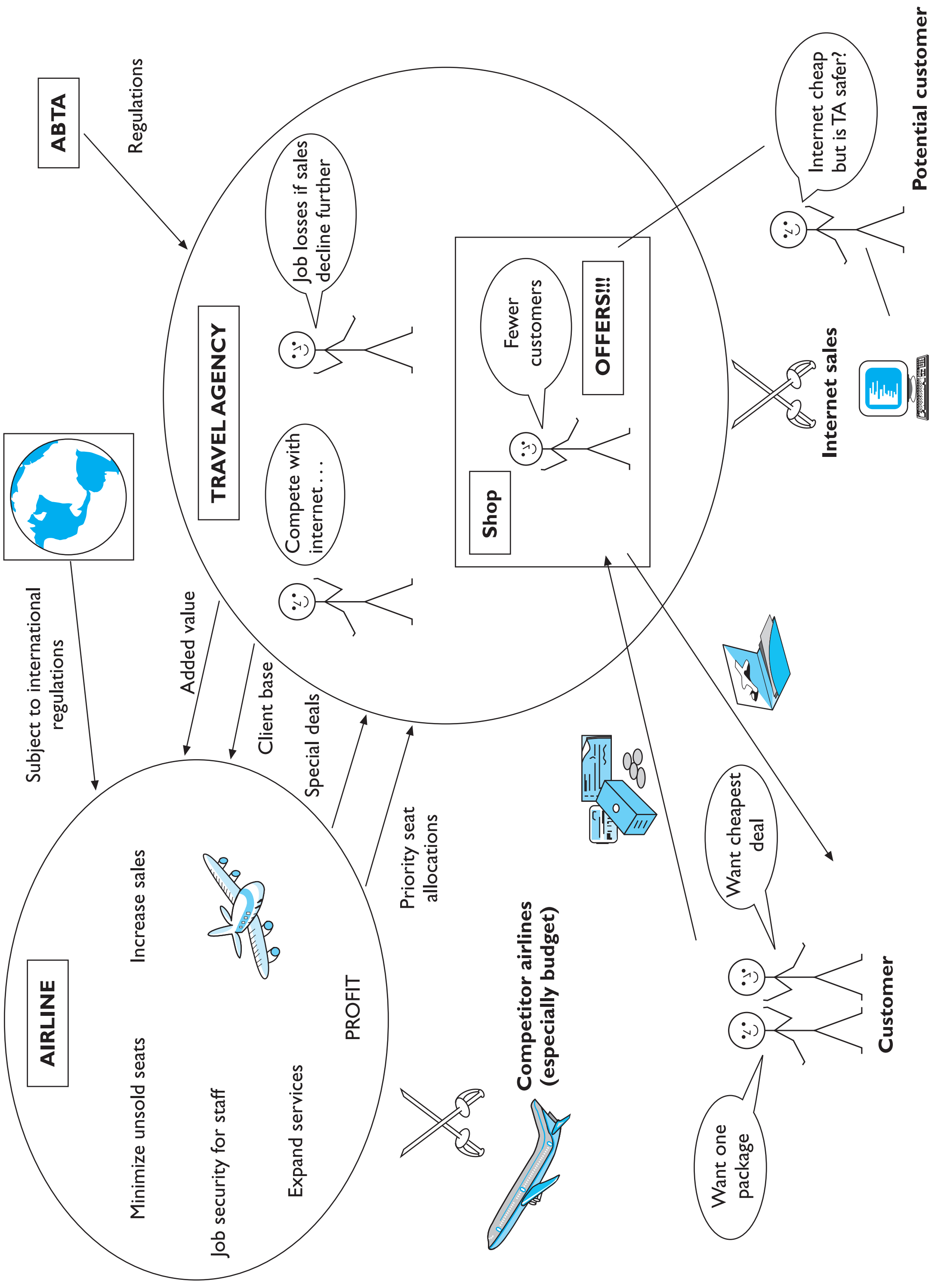


Figure 13.3 A rich picture of a travel agency

design situation. It provides an understandable summary of the designer's understanding that can be easily checked with stakeholders, and it can even be developed collaboratively with stakeholders as part of the consultation process – allowing all parties to contribute to the rich picture sketch. These benefits have led to a number of researchers in HCI proposing their use (outside the full SSM methodology) to inform the design process (see, for example, [239]).

At the next stage in SSM we move from the real world to the systems world and attempt to generate *root definitions* for the system, which define the essence of what the system is about. There may be several root definitions of a system, representing different stakeholder perspectives, for example. Root definitions are described in terms of specific elements, summarized using the acronym, CATWOE:

Clients – those who receive output or benefit from the system.

Actors – those who perform activities within the system.

Transformations – the changes that are effected by the system. This is a critical part of the root definition as it leads to the activities that need to be included in the next stage. These 'transform' the inputs of the system into the required outputs.

Weltanschauung – (from the German) meaning world view. This is how the system is perceived in a particular root definition.

Owner – those to whom the system belongs, to whom it is answerable and who can authorize changes to it.

Environment – the world in which the system operates and by which it is influenced.

Root definition for airline management: an airline booking system

Revisiting our earlier example, an international airline is considering introducing a new booking system for use by associated travel agents to sell flights directly to the public. That is, a system owned by the airline management; operated by associated travel agency staff; working in associated travel agency offices worldwide; operating within regulations specified by international civil aviation authorities and national contract legislation; to sell flights to and reserve seats for customers; and to generate a profit for the company.

- **Client:** customer
- **Actor:** travel agency staff
- **Transformation:** customer's intention and request to travel transformed into sale of seat on flight and profit for organization
- **Weltanschauung:** profits can be optimized by more efficient sales
- **Owner:** airline management
- **Environment:** Regulations of international civil aviation authorities and national contract legislation. Local agency policies worldwide

Once the root definitions have been developed, the *conceptual model* is devised. The conceptual model defines what the system has to do to fulfill the root definitions. It includes identifying the transformations and activities in the system and modeling them hierarchically in terms of what is achieved and how it is achieved. This process is iterative and is likely to take several cycles before it is complete and accurate.

Next we return to the real world with our systems descriptions and compare the actual system with the conceptual model, identifying discrepancies and thereby highlighting any necessary changes or potential problems. For example, a particular activity may have more processes in the real world than in the conceptual model, which may suggest that a reduction of processes for that activity is needed.

In the final stages, we determine which changes are necessary and beneficial to the system as a whole – changes may be structural, procedural or social, for example – and decide on the actions required to effect those changes.

SSM is a flexible approach, which supports detailed consideration of the design context. However, it takes practice to use effectively. There is no single right (or wrong) answer – the SSM is successful if it aids the designer's understanding of the wider system.

13.3.4 Participatory design

Participatory design is a philosophy that encompasses the whole design cycle. It is design in the workplace, where the user is involved not only as an experimental subject or as someone to be consulted when necessary but as a member of the design team. Users are therefore active collaborators in the design process, rather than passive participants whose involvement is entirely governed by the designer. The argument is that users are experts in the work context and a design can only be effective within that context if these experts are allowed to contribute actively to the design process. In addition, introduction of a new system is liable to change the work context and organizational processes, and will only be accepted if these changes are acceptable to the user. Participatory design therefore aims to refine system requirements iteratively through a design process in which the user is actively involved.

Participatory design has three specific characteristics. It aims to improve the work environment and task by the introduction of the design. This makes design and evaluation context or work oriented rather than system oriented. Secondly, it is characterized by collaboration: the user is included in the design team and can contribute to every stage of the design. Finally, the approach is iterative: the design is subject to evaluation and revision at each stage.

The participatory design process utilizes a range of methods to help convey information between the user and designer. They include

Brainstorming This involves all participants in the design pooling ideas. This is informal and relatively unstructured although the process tends to involve 'on-the-fly' structuring of the ideas as they materialize. All information is recorded without judgment. The session provides a range of ideas from which to work. These can be filtered using other techniques.

Storyboarding This has been discussed in more detail in Chapter 6. Storyboards can be used as a means of describing the user's day-to-day activities as well as the potential designs and the impact they will have.

Workshops These can be used to fill in the missing knowledge of both user and designer and provide a more focussed view of the design. They may involve mutual enquiry in which both parties attempt to understand the context of the design from each other's point of view. The designer questions the user about the work environment in which the design is to be used, and the user can query the designer on the technology and capabilities that may be available. This establishes common ground between the user and designer and sets the foundation for the design that is to be produced. The use of role play can also allow both user and designer to step briefly into one another's shoes.

Pencil and paper exercises These allow designs to be talked through and evaluated with very little commitment in terms of resources. Users can 'walk through' typical tasks using paper mock-ups of the system design. This is intended to show up discrepancies between the user's requirements and the actual design as proposed. Such exercises provide a simple and cheap technique for early assessment of models. PICTIVE [242] is one such approach to paper prototyping, which includes representative stakeholders in a video recorded design session. Each participant prepares 'homework' focussing on the requirements of the system from their particular perspective, which is then used to introduce and orientate the PICTIVE session. Materials such as sticky notes, highlighters, plastic labels, paper and scissors are used on a shared design surface to produce a low-tech prototype of the proposed system, which is finally tested by the group against the tasks identified.

Such methods are not exclusively used in participatory design, of course, and can be used more widely to promote clearer understanding between designer and stakeholders.

Participatory design originated in Scandinavia, where it is now promoted by law and in accepted work practices, but it has not been widely practiced, at least in its fullest form, elsewhere. This may be due to the time and cost involved in what is, by definition, a context-specific design, as well as the organizational implications of the shift of power and responsibility.

However, principles from the participatory design approach have been incorporated in a number of widely used methodologies. We outline one of them – to show how a participatory philosophy can be integrated into organizational and stakeholder analysis.

Effective Technical and Human Implementation of Computer-based Systems (ETHICS)

ETHICS [243] is a method developed by Enid Mumford within the socio-technical tradition, but it is distinct in its view of the role of stakeholders in the process. In the

DESIGN FOCUS

Tomorrow's hospital – using participatory design

The nurse walks around the ward to a patient's bedside. She takes her PDA from her pocket and enters the patient's name. Her PDA is connected via a wireless network to the central patient treatment database and the patient's information comes on screen including reminders of treatments needed. First is a blood pressure and heart rate check. She does these checks and enters the results. A mild painkiller is also prescribed. She gets the tablet, which is individually wrapped in a bar coded packet. Her PDA has a built-in bar code reader and she scans this across the tablet. The system registers the drug's use and removes it from the to-do list as she gives the patient the tablet with a glass of water.

A picture of tomorrow's hospital? In fact, this is exactly the day-to-day activity in Hospital da Trofa, just outside Porto, Portugal. There have been numerous attempts to use PDA-based systems in hospitals. But most have failed. So why is this one being used? The hospital is part of a European Commission funded project Team-HOS and the system was designed using a methodology that has a strong *participatory* focus [316]. From the beginning, nurses, doctors, pharmacologists and dieticians were involved in and in control of the design. It has benefited from their knowledge and experience, which is why it does the right things for the context. Furthermore, the pride with which the hospital staff describe the system shows that they really feel it is their system, not one imposed from above.

See the book website for an extended case study: /e3/casestudy/trofa/

See: www.ieeta.pt/team-hos/ Source: Courtesy of Professor J. Artur Vale Serrano

ETHICS methodology, stakeholders are included as participants in the decision-making process. ETHICS considers the process of system development as one of managing change: conflicts will occur and must be negotiated to ensure acceptance and satisfaction with the system. If any party is excluded from the decision-making process then their knowledge and contribution is not utilized and they are more likely to be dissatisfied. However, participation is not always complete. Mumford recognizes three levels of participation:

Consultative – the weakest form of participation where participants are asked for their opinions but are not decision makers.

Representative – a representative of the participant group is involved in the decision-making process.

Consensus – all stakeholders are included in the decision-making process.

The usual practice is that design groups are set up to include representatives from each stakeholder group and these groups make the design decisions, overseen by a steering committee of management and employee representatives. The design groups then address the following issues and activities:

1. *Make the case for change*. Change for its own sake is inappropriate. If a case cannot be made for changing the current situation then the process ends and the system remains as it is.
2. *Identify system boundaries*. This focusses on the context of the current system and its interactions with other systems, in terms of business, existing technology, and internal and external organizational elements. How will the change impact upon each of these?
3. *Describe the existing system*, including a full analysis of inputs and outputs and the various other activities supported, such as operations, control and coordination.
4. *Define key objectives*, identifying the purpose and function of each area of the organization.
5. *Define key tasks*: what tasks need to be performed to meet these objectives?
6. *Define key information needs*, including those identified by analysis of the existing system and those highlighted by definition of key tasks.
7. *Diagnose efficiency needs*, those elements in the system that cause it to underperform or perform incorrectly. If these are internal they can be redesigned out of the new system; if they are external then the new system must be designed to cope with them.
8. *Diagnose job satisfaction needs*, with a view to increasing job satisfaction where it is low.
9. *Analyze likely future changes*, whether in technology, external constraints (such as legal requirements), economic climate or stakeholder attitudes. This is necessary to ensure that the system is flexible enough to cope with change.
10. *Specify and prioritize objectives based on efficiency*, job satisfaction and future needs. All stakeholders should be able to contribute here as it is a critical stage and conflicting priorities need to be negotiated. Objectives are grouped as either primary (must be met) or secondary (desirable to meet).

The final stages of the ETHICS approach focus on the actual design and evaluation of the system. Necessary organizational changes are designed alongside the technical system. These are then specified in detail, implemented and evaluated.

The ETHICS approach attempts to reach a solution that meets both user and task requirements by having specialist teams negotiate objectives and rank potential solutions. The emphasis is on reaching a solution that ranks highly on job satisfaction to ensure that the solution is acceptable.

It ensures participation at every stage through representative design teams, but can be expensive and time consuming to carry out. However, the benefits in terms of job satisfaction and higher productivity may balance out the initial investment.

13.3.5 Ethnographic methods

All of the approaches considered so far have included some level of consultation and observation of the stakeholders. However, the focus of this has been on gathering stakeholder perspectives rather than specifically studying actual work practice. It can be argued that work can only be understood and studied in context. This is consonant with the ideas of distributed cognition. Taking a worker away from the workplace changes the very nature of the worker's actions. Real action is *situated action*; it occurs in interaction with the materials and people of the workplace [334]. In extremis, it is claimed that an action can only be understood in the place, in the social situation, and at the *time* at which it occurred. Such a level of contextualization is obviously useless for design, and its advocates will in practice generalize from their observations, even if they ostensibly eschew such generalization.

Even if one does not wish to take such an extreme view, it is clear that studying workers in their own situations is extremely worthwhile.

Many branches of sociology and anthropology have long recognized that one cannot study people divorced from their social and cultural context. In particular, *ethnography* has become very influential, particularly in the study of group systems. We were introduced to ethnography briefly in Chapter 9 but here we will consider it in a little more detail, and discuss an approach to contextual requirements capture that reflects the ethnographic tradition but has been developed to be highly practical.

Ethnography is based on very detailed recording of the interactions between people and between people and their environment. It has a special focus on social relationships and how they affect the nature of work. The ethnographer does not enter actively into the situation, and does not see things from a particular person's viewpoint. However, an aim is to be encultured, to understand the situation from within its own cultural framework. Culture here means that of the particular work-group or organization, rather than that of society as a whole. Ethnographers try to take an unbiased and open-ended view of the situation. They report and do not like to speculate, so it is often unclear how well their approach can contribute to the design of new systems.

Ethnography and participatory design

The ethnographic approach differs markedly from the approach of participatory design. In participatory design the workers come *out* of their work situation, either physically or mentally, and share the design task with the professional designers – effectively the workers become designers. The participatory designer enters into the subjective experience of the workplace. Ethnographic and other situated approaches take the analyst *into* the workplace, while retaining a level of objectivity. The advantage is that the analyst sees the whole group's perspective, rather than that of involved individuals, but the analyst, however much in tune with the workers, is still 'out there'. On the other hand, involving the workers in the design process in itself increases their motivation and acceptance whether or not the resulting design is 'optimal'.

We now look briefly at an approach that takes a pragmatic rather than purist ethnographic perspective, *contextual inquiry*.

Contextual inquiry

Contextual inquiry has much in common with the ethnographic tradition: it studies the user in context, trying to capture the reality of his work culture and practice. However, it is also an approach rooted in practice and it differs in a number of significant ways from pure ethnographic study: the intention is to understand and to interpret the data gathered, and rather than attempting to take an open-ended view, the investigator acknowledges and challenges her particular focus. In addition, the explicit aim is to design a new system, whereas in a pure ethnographic study, it would be open ended.

The model of contextual inquiry is of the investigator being apprenticed to the user to learn about his work. Interviews take place in the workplace so that the objects, artifacts and relationships of the work can be better understood. Examples of work are collected and both verbal and non-verbal communication is studied. The idea is to be as comprehensive in the data gathering as possible and to be concrete. Another central notion of contextual inquiry is that of partnership: the user is the expert in the workplace and is therefore encouraged to lead the investigation. However, the investigator is not a passive observer. Her objective is to gain a shared understanding of how the work happens and, to do so, she questions meaning and offers interpretations of what she observes. The aim is to draw out the implications of comments and actions and understand (rather than assume) what they really mean. In order to do this honestly and effectively the investigator must know her focus – her pre-existing beliefs and assumptions about the situation – and be prepared to challenge and adjust them in the face of new information.

Contextual inquiry focusses on a 2–3 hour interview with the user in the workplace. The idea is to capture and record as much detail as possible, including what the user says and does (step by step), how he communicates and coordinates with others, his feelings and responses to the situation, and a shared understanding of the meaning of actions and artifacts. In addition, objects, examples and artifacts of work are collected and annotated, and the physical work environment is sketched and annotated to show how it is used.

A number of models of the work are developed to capture what is important in the user's work situation:

- The sequence model elaborates the steps required to complete a specific task, as well as the triggers that initiate that sequence of steps.
- The physical model maps the physical work environment and how it impacts upon work practice, for example, an office plan showing where different work activities happen.
- The flow model shows the lines of coordination and communication between the user and other participants within and outside the workplace.
- The cultural model reflects the influences of work culture and policy and shows the scope of these influences. This may include official or unofficial codes of behavior, common expectations (which may or may not be explicit) and value systems.
- The artifact model describes the structure and use of a particular artifact within the work process.

Once the interview is complete, the data must be consolidated with those from other users and stakeholders. There is not space here to elaborate the whole process (see [35] for details) but interviews are transcribed and interpreted as soon as possible after the event by the design team. The team comes together to consider the interview data and to identify commonalities across stakeholders. Affinity diagrams are used to group related information by posting notes on the wall representing a particular comment or observation and grouping these into a hierarchy of related themes. The themes are not pre-determined but arise from the data as they are examined.

Each of the models above is also consolidated across users to provide a common view of the situation. The result is a representation of the required task sequences, artifacts and communication channels that must be supported in the new system as well as the physical and cultural constraints that must be taken into account.

13.4 SUMMARY

We have seen that organizational factors can make or break the deployment of information and communications technology. Any computer system may interfere with the existing authority and social relationships within an organization. There may be a mismatch between those who benefit, and those who do the work. Even where there is no in-built bias, the free rider may put in little personal effort, benefiting from the work of the rest of the group.

We need to identify stakeholders who will be affected directly or indirectly. In particular, junior and middle management may feel they lose control and authority by the introduction of electronic communications. Another example is teleworking, which is made easier by advances in telecommunications, but which makes the worker less visible to management. Even where a system is perceived to be useful, it is hard to quantify its benefits as they are diffused throughout the organization.

In this chapter, we have discussed a selection of socio-organizational approaches to capturing stakeholder requirements, including socio-technical models, soft systems methodology, participatory design and ethnographic approaches. Socio-technical models focus on representing both the human and technical sides of the system in parallel to reach a solution that is compatible with each. SSM models the organization, of which the user is part, as a system. Participatory design sees the user as active not only in using the technology but in designing it. Ethnography, on the other hand, views the user in context, attempting to collect an unbiased view of the user's work culture and practice.

Exercises

13.1 A group of universities has decided to collaborate to produce an information system to help potential students find appropriate courses. The system will be distributed free to schools and careers offices on CD-ROM and will provide information about course contents and requirements, university and local facilities, fees and admissions procedures. Identify the main stakeholders for this system, categorize them and describe them and their activities, currently and with regard to the proposed system, using the CUSTOM framework.

13.2 For the scenario proposed above:

(i) Produce a rich picture showing the problem situation (you can use any format that you find helpful).

(ii) Produce a root definition, using CATWOE, of the system from the viewpoint of the university.

What transformations or activities are required to make sure that the root definition is supported?

13.3 The example in Section 13.3.2 (soft systems methodology) provides a root definition for an airline booking system from the perspective of the airline owner. How would this change if it was presented from the perspective of the customer?

13.4 Find case studies of participatory design in action and use these to provide a critique of the approach. What are the benefits and weaknesses of participatory design and how might any weaknesses be addressed?

13.5 You are designing a new system to help people manage their 'to do' lists. Use the contextual inquiry approach to interview a colleague to see how they make use of such lists. Make sure you interview them in context – in their study or workplace for example. Produce sequence, flow, artifact, cultural and physical models of the activity.

RECOMMENDED READING

K. D. Eason, *Information Technology and Organizational Change*, Taylor and Francis, 1988.
Clear coverage of socio-technical and organizational issues in design.

P. B. Checkland, *Systems Thinking, Systems Practice: includes a 30 Year Retrospective*, John Wiley, 1999.
The classic text on soft systems methodology in detail, in a new edition with reflections on the development and use of the methodology.

J. Greenbaum and M. Kyng, *Design at Work: Cooperative Design of Computer Systems*, Lawrence Erlbaum, 1991.
An excellent collection of papers on participatory design philosophy and practice.

E. Mumford, *Effective Systems Design and Requirements Analysis: The ETHICS Approach*, Palgrave, 1995.
A book outlining the ETHICS approach by its developer.

H. Beyer and K. Holtzblatt, *Contextual Design: Defining Customer Centered Systems,* Morgan Kaufmann, 1998.
A full exposition of the contextual design methodology, of which contextual inquiry is a part.

14 Communication and collaboration models

OVERVIEW

All computer systems, single-user or multi-user, interact with the work-groups and organizations in which they are used.

- We need to understand normal human–human communication:
  - face-to-face communication involves eyes, face and body
  - conversation can be analyzed to establish its detailed structure.
- This can then be applied to text-based conversation, which has:
  - reduced feedback for confirmation
  - less context to disambiguate utterances
  - slower pace of interaction

  but is more easily reviewed.
- Group working is more complex than that of a single person:
  - it is influenced by the physical environment
  - experiments are more difficult to control and record
  - field studies must take into account the social situation.

14.1 INTRODUCTION

No man is an *Iland*, intire of it selfe; every man is a peece of the *Continent*, a part of the *maine*.

John Donne[1]

It is clear that groupware systems, such as email or conferencing systems, involve more than one person. However, to some extent all systems influence and are influenced by the groups and social situations in which they are placed. The field of computer-supported cooperative work (CSCW) encompasses both specific groupware systems and the effects of computers on cooperative working in general.

We begin this chapter by looking at human communication. Effective communication clearly underlies much collaborative work and many systems aim to support communication at a distance. Face-to-face communication is often seen as the ideal to which computer-mediated communication should aim. Section 14.2 describes some of its features, and shows how even video-based communications lose many of the subtle cues. Body language, tone of voice and eye contact are all crucial in enabling smooth conversation. We then look at a slightly higher level in Section 14.3, at the structure of conversation. In particular, we will see that typical utterances are ambiguous and dependent on the context in which they are spoken. This section concludes by looking at speech act theory and the Coordinator system which has been both influential and controversial in CSCW. Understanding spoken conversation gives some clues as to the effective design of text-based communication such as email systems or electronic conferencing. Section 14.4 discusses text-based communication in this context, including a discussion of the relative merits and features of linear text and hypertext systems.

In Section 14.5 we move away from computer-mediated communication and look at the wider issues of group working. Groups are dynamic both in composition and behavior, and effective group working is dependent on the work environment. Because of these and other factors, the study of group behavior, and therefore evaluating groupware, is far more complex than that of single-user systems.

14.2 FACE-TO-FACE COMMUNICATION

Face-to-face contact is the most primitive form of communication – primitive, that is, in terms of technology. If, on the other hand, we consider the style of communication, the interplay between different channels and productivity, we instead find that face-to-face is the most sophisticated communication mechanism available.

1 Devotions upon Emergent Occasions, XVII, 1624.

The first thing to note is that face-to-face communication involves not just speech and hearing, but also the subtle use of body language and eyegaze. We will discuss a range of these phenomena, and how they influence our use of computer-mediated communications. We will concentrate on two-person conversations as group dynamics are discussed later in Section 14.5. Also, we will principally compare face-to-face with video and audio channels and we will address the special problems of text-based communications in Section 14.4.

14.2.1 Transfer effects and personal space

When we come to use computer-mediated forms of communication, we carry forward all our expectations and social norms from face-to-face communication. People are very adaptable and can learn new norms to go with new media (for example, the use of 'over' for turn-taking when using a walkie-talkie). However, success with new media is often dependent on whether the participants can use their existing norms. Furthermore, the rules of face-to-face conversation are not conscious, so, when they are broken, we do not always recognize the true problem. We may just have a feeling of unease, or we may feel that our colleague has been rude.

An example of these problems concerns personal space. When we converse with one another we tend to stand with our heads a fairly constant distance apart. If people start to converse at opposite ends of a room, they will quickly move toward one another until they are a few feet apart. The exact distance depends somewhat on context; a high level of noise may make people come closer just to be heard. However, even in crowded rooms, conversants will dip their heads toward one another whilst speaking and then straighten up to restore their personal distance. Direction is also important. We can accept people closer to us if they are at our sides or behind than if we are facing them. Because of this, passengers on tube trains, forced to be close, will incline their faces at an angle to one another whilst talking.

Personal space also differs across cultures: North Americans get closer than Britons, and southern Europeans and Arabs closer still. This can cause considerable problems during cross-cultural meetings. Imagine a Briton, Eustace Warbuck-Smyth, and an American, Bud Sterton, conversing. After a few minutes Eustace is bent backwards over a table, trying to maintain his personal distance, whilst Bud stands almost knee to knee trying to get close enough. Eustace feels Bud is either rather aggressive, or possibly over-friendly. Bud, on the other hand, feels Eustace is rather distant and uninterested. Unless the situation gets extreme, or the participants are trained in non-verbal skills, they will be unaware of why they feel uncomfortable.

A similar problem can occur in a video conference. Imagine Eustace and Bud have monitors with cameras mounted above, so that their offices are connected. The zoom on each camera is adjustable and Bud's camera is set with a wide focus, whilst Eustace's is set with a high level of zoom. So, if Bud and Eustace are the same distance from their cameras and monitors, then Bud sees Eustace's whole face filling the screen, whereas Eustace sees Bud sat on his chair in the middle of his office. Eustace moves closer to the monitor to see Bud more clearly, while Bud pushes his chair back to get away from Eustace's 2 foot (60 cm) high face – *touché*. Of course, the problem gets worse if the cameras are positioned in different places relative to the monitors, or if the monitors are different sizes. Ideally, Bud ought to be able to adjust the zoom on Eustace's camera and vice versa. In fact, there is some evidence that the 'glass wall' afforded by the video screen makes the precise distance less important, which could have a positive effect during cross-cultural meetings. However, gross distortions, as described above, need to be avoided.

14.2.2 Eye contact and gaze

Long-term gazing into one another's eyes is usually reserved for lovers. However, normal conversation uses eye contact extensively, if not as intently. Our eyes tell us whether our colleague is listening or not; they can convey interest, confusion or boredom. Sporadic direct eye contact (both looking at one another's eyes) is important in establishing a sense of engagement and social presence. People who look away when you look at them may seem shifty and appear to be hiding something. Furthermore, relative frequency of eye contact and who 'gives way' from direct eye contact is closely linked to authority and power.

Naturally, all these clues are lost if we have no visual contact. However, the misleading clues via a video connection can be worse. In Chapter 19 (Section 19.3.4) we discuss the problems of obtaining effective eye contact with standard video equipment. If the camera is strapped to the top of the monitor (a common arrangement) both participants will look as if their eyes are slightly dropped. We will look at some technical solutions to this in Chapter 19.

Despite these problems with direct eye contact, many signals can be easily read through a video channel. You can see whether your colleague looks quizzical or bored, confused or excited. This involves not just the eyes, but the whole facial expression, and this is apparent even on poor-quality video or very small (pocket-TV-sized) monitors. Experiments have shown that remotely working participants experience a greater sense of social presence if video is used in addition to an audio link.

As well as having a role in establishing rapport between the participants, eyegaze is useful in establishing the focus of the conversation. If you say 'now where does this screw go?', there may be many screws, but your colleague can see which one you are looking at. Video connections are unlikely to show enough of your office for your colleague to be able to interpret such clues, but a focus that just catches the corner of the monitor and desk can help.

14.2.3 Gestures and body language

In a similar but more direct way, we use our hands to indicate items of interest. This may be conscious and deliberate as we point to the item, or may be a slight wave of the hand or alignment of the body. Again, a video connection may not be sufficient to allow our colleagues to read our movements. This can be a serious problem since our conversation is full of expressions such as ‘let’s move this one there’, where the ‘this’ and ‘there’ are indicated by gestures (or eyegaze). This is called *deictic reference* (see Chapter 19, Sections 19.4 and 19.6 for more details).

Several groupware systems attempt to compensate for these losses. In Section 19.4.2, we discuss the idea of a group pointer, a mouse-controlled icon which can be used to point to things on a shared screen. Somewhat more esoteric, but more immediate, are the shared work surfaces (Section 19.4.3) which mix an image of the participants’ hands with the electronic screen. The participants can then simply point at the relevant item on the screen, as they would face-to-face.

Of course, the group pointer, although used in remote groupware, is also used in synchronous co-located groupware such as meeting rooms. That is, even though the participants can converse face-to-face, they still need deictic aids. One reason for this is that their electronic screens, although logically shared (they can all see the same thing), are not physically shared (they are in different places). So, if Jemima points to her own screen, her colleagues do not know what she is pointing at.

Even when the participants are in the same room, the existence of electronic equipment can interfere with the body language used in normal face-to-face communication. The fact that attention is focused on keyboard and screen can reduce the opportunities for eye contact. Also, large monitors may block participants’ views of one another’s bodies, reducing their ability to interpret gestures and body position. Most computer-supported meeting rooms recess monitors into the desks to reduce these problems.

14.2.4 Back channels, confirmation and interruption

It is easy to think of conversation as a sequence of utterances: A says something, then B says something, then back to A. This process is called *turn-taking* and is one of the fundamental structures of conversation. However, each utterance is itself the result of intricate negotiation and interaction. Consider the following transcript:

Alison:	Do you fancy that film . . . *er* . . . ‘The Green’ *um* . . . it starts at eight.
Brian:	Great!

Alison has asked Brian whether he wants to go to the cinema (or possibly to watch the television at home). She is a bit vague about the film, but Brian obviously does not mind! However, if we had listened to the conversation more closely and watched Alison and Brian we would have seen more exchanges. As Alison says ‘that film *er* . . .’, she looks at Brian. From the quizzical look on his face he obviously does not know which film she is talking about. She begins to expand ‘The Green *um* . . .’, and light dawns; she can see it in his eyes and he probably makes a small affirmative sound ‘uh huh’.

The nods, grimaces, shrugs of the shoulder and small noises are called *back channels*. They feed information back from the listener to the speaker at a level below the turn-taking of the conversation. The existence of back channels means that the speaker can afford to be slightly vague, adding details until it is obvious that the listener understands. Imagine making no response as someone talks to you, no little 'yes'es, no nods or raised eyebrows. You could answer questions and speak in turn, but not use back channels. It is likely that your colleague would soon become very uncomfortable, possibly rambling on with ever more detailed explanations, looking for some sign of understanding:[2]

> Do you fancy that film . . . *er* . . . 'The Green' *um* . . . the one with Charles Dermot in . . . you know with that song, *er* and the black cat on the poster . . . *uhh*

These back channel responses use a range of sensory channels. So, as we restrict the forms of communication we lose the back channels. Even video communications tend to use, at most, head and shoulder shots, so we lose some body movement and gestures. On the other hand, a larger view means reduced detail, so we lose information whatever focus we choose. Audio-only links (such as the telephone) have to rely on purely verbal back channel responses – the little 'yes'es. Surprisingly, despite the loss of many back channels, people still cope well with these restricted media, and communication is still reasonably effective. However, you may have had the experience, when speaking to someone on the telephone, of suddenly getting the feeling that they have gone away, or the line has gone dead. This is likely to be when you have received insufficient back channel responses (and perhaps you have been going on a bit). Transcontinental telephones are especially problematic as they are often only half duplex, that is the sound only goes in one direction at a time. So, while you are speaking, you can hear none of your partner's back channel responses.

Text-based communication, in electronic conferencing, usually has no back channels whatsoever. Any confirmation must be given explicitly in the listener's next utterance. This may confuse an analysis of text-based conversation as the utterances do not correspond simply to utterances in speech.

14.2.5 Turn-taking

As well as giving confirmation to the speaker that you understand, and indications when you do not, back channels can be used to interrupt politely. Starting to speak in the middle of someone's utterance can be rude, but one can say something like 'well uh' accompanied by a slight raising of the hands and a general tensing of the body and screwing of the eyes. This tells the speaker that you would like to interrupt, allowing a graceful transition. In this case, the listener *requested* the floor. *Turn-taking* is the process by which the roles of speaker and listener are exchanged. Back channels are often a crucial part of this process.

2 Don't try this as an experiment on your friends, or you may end up without any! Instead try it with a colleague who knows what is going on. Even when you both know not to expect back channel responses the experience can be disconcerting. Furthermore, you will both find it very difficult to refrain from back channel responses.

DESIGN FOCUS

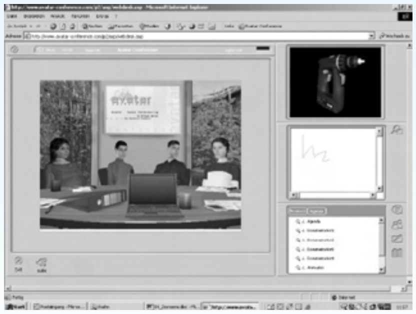
Looking real – Avatar Conference

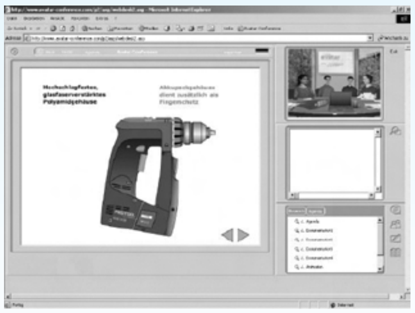
Avatars, artificial representations of people, are used in various forms of online communication. Often these are based on some form of cartoon image, but the Avatar Conference project has produced a virtual meeting environment with photo-realistic avatars. These are produced by photographing participants from several angles and then using the photographs to texture map an artificial figure. This is then rendered in a 3D virtual meeting room.

Participants can use internet phone or simple text chat and the latter has translation options to aid international meetings. The avatar for the current speaker moves its mouth and makes other animations, but is not synchronized with the actual speech. The aim is to make the participants more engaged in the online meeting.

The screen shots show two views of the conference application. One with the virtual meeting room central (left), the other with the currently displayed presentation materials central (right). People found the 3D avatars exciting and an initial 'hook' – this is very important, remember in the Introduction, the need for a system to be actually *used*.

However, the participant response after using the application for even a short time was in favor of the more presentation-centered rather than avatar-centered view. It may be that current avatars are not quite realistic enough to warrant the central position, or it may reflect the fact that in a real meeting the focus is on the presentation rather than on the other participants.





See: www.exodus.gr/Avatar_Conference/ and www.avatame.com/ Screen shot frame reprinted by permission from Microsoft Corporation. Screen shots courtesy of AVATAR-Conference project team

In other cases, the speaker may explicitly *offer* the floor to the other participant. This may be in the form of a direct question, 'what do you think?', or simply a very strong change of tone.

More often the speaker will offer the floor to the listener by leaving a small gap in speech. These gaps are typically no more than a fraction of a second; indeed gaps of even a few seconds give completely different signals and interrupt the flow of conversation. The gap is often at some point which may require clarification, or where the listener may want to comment. So, Alison may well have waited for half a second after saying 'Do you fancy that film', in case Brian were to respond 'Oh you mean the one with the black cat on the poster'. As he did not say anything, she continued with her turn.

The role of '*um*'s and '*ah*'s is very important. They can be used by either participant during the gap to *claim* the turn. So, if Brian wanted to respond in the middle of the utterance, but had not yet framed his utterance, he might begin '*um* the one . . .'. As it was, Brian did not respond, so Alison starts '*er*' which says to Brian 'I'm going to continue, but I'm thinking'. Alternatively, Alison could have started to '*er*' as soon as she had said the word 'film'. This would have told Brian *not* to interrupt.

These turn-offering gaps are just the places where the speaker expects some back channel response even if no turn exchange takes place. A total lack of response will be taken, depending on the circumstances, as assent to the speaker, or perhaps as lack of understanding.

As we can see, turn-offering gaps form a central part in the eliciting of back channel responses and in negotiating turn-taking. They are obviously connected principally with the audio channel (although some gestures may be used to maintain or claim the floor). Half-duplex channels (such as intercontinental phone calls) are volume sensitive in order to track the speaker. Unfortunately, some of the '*um*'s and '*er*'s used to maintain or claim the floor may fall below the volume threshold and thus not be transmitted. This may lead to apparent rude interruptions or snubbing of one party by the other.

An even more serious problem is encountered during long-distance, satellite-based communications due to the time lags. To transmit a signal, it must go up to the satellite and then back down to the earth. A geostationary satellite is at a height of approximately 100,000 km above the earth – a quarter of the distance to the moon. Radio waves will take about 700 milliseconds to go up to the satellite and back down again. This time, together with the processing delays on the ground and in the satellite, add up to about a 2 second lag. There is thus a 4 second gap between one participant doing or saying something and when the effects of that upon the other participant become evident.

We now imagine Alison and Brian talking via satellite. Alison pauses for half a second after the words 'that film', 2 seconds later Brian hears the end of the word 'film', after a few hundred milliseconds he notices the gap and begins to say 'Oh'; by this time Alison has waited $2^1/_2$ seconds, and the gap is getting embarrassing, so she continues 'The Green *um* . . . the one with Charles Dermot in'. Then, 2 seconds into this, that is over 4 seconds since she began to pause, she hears Brian try to cut in (how rude!), but he stops talking again when he hears her continuing.

The above scenario is *not* contrived. Tapes of video conferences show just this behavior, with a single speaker going on and on as all her attempts to pass on the floor fail. There is no obvious solution to this problem, except the technological one of using high-bandwidth land or sub-ocean lines, when these become available, rather than satellite.

14.3 CONVERSATION

We have looked at the low-level issues of speech and gesture during face-to-face conversation. We now turn to the structure of the conversation itself. Most analysis of conversation focusses on two-person conversations, but this can range from informal social chat over the telephone to formal courtroom cross-examination. As well as the discipline of *conversational analysis*, there are other sociological and psychological understandings of conversation. However, the techniques, as 'borrowed' and used to study computer-mediated conversation, would not always find favor with the purist from the discipline from which they originated!

There are three uses for theories of conversation in CSCW. First, they can be used to analyze transcripts, for example from an electronic conference. This can help us to understand how well the participants are coping with electronic communication. Secondly, they can be used as a guide for design decisions – an understanding of normal human–human conversation can help avoid blunders in the design of electronic media. Thirdly, and most controversially, they can be used to drive design – structuring the system around the theory.

We will concentrate mainly on the first goal, although this will have implications throughout for design. Only when we consider *speech act theory* and *Coordinator* in Section 14.3.6 will we see an example of a theory-driven system.

14.3.1 Basic conversational structure

Imagine we have a transcript of a conversation, recalling from Chapter 9 that the production of such a transcript is not a simple task. For example, a slightly different version of Alison and Brian's conversation may look like this:

Alison: Do you fancy that film?
Brian: The *uh* (*500 ms*) with the black cat – 'The Green whatsit'?
Alison: Yeah, go at *uh* . . . (*looks at watch – 1.2 s*) . . . 20 to?
Brian: Sure.

This transcript is quite heavily annotated with the lengths of pauses and even Alison's action of looking at her watch. However, it certainly lacks the wealth of gesture and back channel activity that were present during the actual conversation. Transcripts may be less well documented, perhaps dropping the pause timings, or more detailed, adding more actions, where people were looking and some back channelling. Whilst thinking about the structure of conversation, the transcript above is sufficient.

As we have noted previously, the most basic conversational structure is *turn-taking*. On the whole we have an alternating pattern: Alison says something, then Brian, then Alison again. The speech within each turn is called an *utterance*. There can be exceptions to this turn-taking structure even within two-party conversation. For example, if there is a gap in the conversation, the same party may pick up the thread, even if she was the last speaker. However, such gaps are normally of short duration, enough to allow turn-claiming if required, but short enough to consider the speech a single utterance.

Often we can group the utterances of the conversation into pairs: a question and an answer, a statement and an agreement. The answer or response will normally follow directly after the question or statement and so these are called *adjacency pairs*. We can look at Alison and Brian's conversation above as two adjacency pairs, one after the other. First, Alison asks Brian whether he knows about the film and he responds. Second, she suggests a time to go and he agrees. We can codify this structure as: A-x, B-x, A-y, B-y, where the first letter denotes the speaker (Alison or Brian) and the second letter labels the adjacency pair.

The requirement of adjacency can be broken if the pair is interposed with other pairs for clarification, etc.:

Brian:	Do you want some gateau?
Alison:	Is it very fattening?
Brian:	Yes, very.
Alison:	And lots of chocolate?
Brian:	Masses.
Alison:	I'll have a big slice then.

This conversation can be denoted: B-x, A-y, B-y, A-z, B-z, A-x. Adjacency pair 'x' ('Do you want some gateau?'–'I'll have a big slice then') is split by two other pairs 'y' and 'z'. One would normally expect the interposed pairs to be relevant to the outer pair, seeking clarification or determining information needed for the response.

Some would say that the adjacency pair is not just a basic structure of conversation but *the* fundamental structure. It is clearly true that we normally respond to the most recent utterance. However, it is less clear whether a simple pairing up of utterances is always possible or useful.

For an example of this difficulty, let us look back to the transcript at the beginning of the section. We see that the pair structure is not completely clear. Alison's second utterance begins a new pair, 'go at 20 to?', but it began with 'yeah' responding to Brian's previous utterance. Indeed, Brian's first response 'the *uh* . . . with the black cat . . .' could be seen as a request for clarification. That is, we are now looking at the conversation as having a structure of A-x, B-y, A-yz, B-z, where Alison's second utterance serves as both a response to Brian's request for clarification ('y') and starts a new pair concerning the time ('z'). But in this case, what happens to the second half of the original 'x' pair? We are forced to regard it as implicit in one of Brian's utterances. Alison's 'go at 20 to?' clearly suggests that Brian has committed himself, so we assume that the tone of Brian's description of the film suggested acceptance. So, Brian's first utterance, like Alison's second, serves a dual purpose: A-x, B-xy, A-yz, B-z.

Despite these difficulties, we see that the search for adjacency pairs forces us to examine closely the structure of the conversation. Whether such structures are really part of the conversation, or imposed by us upon it, is less clear. Later we shall see far more complex conversational structures.

14.3.2 Context

Take a single utterance from a conversation, and it will usually be highly ambiguous if not meaningless: 'the *uh* with the black cat – "The Green whatsit"'. Each utterance and each fragment of conversation is heavily dependent on *context*, which must be used to *disambiguate* the utterance. We can identify two types of context within conversation:

internal context – dependence on earlier utterances. For example, when Brian says 'masses' in the last transcript, this is meaningful in the light of Alison's question 'and lots of chocolate?'. This in turn is interpreted in the context of Brian's original offer of gateau.

external context – dependence on the environment. For example, if Brian had said simply 'do you want one?', this could have meant a slice of gateau, or, if he had been holding a bottle, a glass of wine, or, if accompanied by a clenched fist, a punch on the nose.

Arguably, even a complete conversation is heavily context dependent – without knowing the situation and the social relations between the participants, how can we understand their words? Taking a more pragmatic approach, the importance of external context has implications for system design and for data collection. From a design perspective, we will look for groupware which both maximizes shared context and which makes the level of sharing clear.

Turning to data collection, we can see the importance of annotating transcripts with gestures, eyegaze and details of the environment. However, if one noted everything down, 90% would be irrelevant to the conversational level of analysis. Recording the interesting details without flooding the transcript is clearly a skilled job. If one has rich recordings, say from several video sources, then a simple verbal transcript may be sufficient as it will be possible to refer back to the video when interesting incidents are found in the written transcript.

When collecting data from groupware systems, it is also very important to have synchronized records of the participants' conversation (whether audio, video or text based) and their electronic workspaces. We need to know what the participants can see on their screens in order to interpret their remarks to one another. In the case where participants may have different views at the same time, we are likely to see *breakdowns* in the conversation, where one participant makes an utterance depending on his screen, whereas his colleague sees something different on her screen.

A specific form of context dependence is *deictic reference* (Section 14.2.3). When accompanied by a pointed finger, an expression like 'that post is leaning a bit' is clearly dependent on external context. However, there are very similar uses of internal context:

Brian:	(*Points*) That post is leaning a bit.
Alison:	That's the one you put in.

Brian's utterance uses external context, whereas Alison's very similar utterance uses internal context. Her 'that' refers to the post Brian was talking about, not the one he is pointing at. To see this, consider the similar fragment:

Brian: The corner post is leaning a bit.
Alison: That's the one you put in.

Real speech, probably more than the written word, is full of *indexicals*, words like 'that', 'this', 'he', 'she' and 'it'. Obviously when used in written text, like *this*, words such as *these* make use of purely internal context. In spoken speech any of the above words can be accompanied by gestures or eyegaze for external context, or simply used, as Alison did, to refer to previous things in the conversation. Some of the words tend to be more likely to be external ('that', 'this') than others ('he', 'she'), but you can easily think of cases of both forms of use. Furthermore, the attachment of pronouns and other indexicals to the things they denote may depend on the semantics of a sentence: 'Oh no! Eustace has hit Bud. He'll kill him, I know he will.' Does the speaker mean that Eustace will kill Bud, or vice versa? The answer depends on the speaker's knowledge of Eustace and Bud. If Bud is a 22 stone (138 kg) trucker and Eustace has trouble lifting cans of beans then we interpret the sentence one way. If, on the other hand, Eustace has a black belt in karate . . .

One consequence of the use of context in speech is the fragmentary nature of utterances. The example transcripts are, if anything, atypically grammatical. Although there is evidence of rules of grammar for the spoken word, these are very different, and much more relaxed, than the written equivalent.

14.3.3 Topics, focus and forms of utterance

Given that conversation is so dependent on context, it is important that the participants have a shared focus. We have addressed this in terms of the external focus – the objects that are visible to the participants – but it is also true of the internal focus of the conversation.

Alison: Oh, look at your roses . . .
Brian: Mmm, but I've had trouble with greenfly.
Alison: They're the symbol of the English summer.
Brian: Greenfly?
Alison: No roses silly!

Alison began the conversation with the *topic* of roses. Brian shifts to the related, but distinct, topic of greenfly. However, for some reason Alison has missed this shift in focus, so when she makes her second utterance, her focus and Brian's differ, leading to the *breakdown* in communication. The last two utterances are a recovery which re-establishes a shared *dialog focus*.

In general, we can go through a transcript annotating the utterances by the topics to which they refer. The identification of topics and assigning utterances to them is a somewhat subjective affair, and one may want to use several levels of topic categorization. Of course, those points where such a labeling is difficult are interesting in themselves. They may either represent potential points of breakdown (as above), or show where external context is needed to disambiguate the conversation. Also of interest is the way that the participants negotiate changes in dialog focus, either because they recognize a divergence, or because one party wants to shift the focus of the conversation.

This sort of analysis can be pursued for its own sake, but has a more pragmatic interest in the analysis of computer-mediated conversation and design of groupware. We want to know where breakdowns occur in order to see whether these are due to the electronic medium. We also want to understand the shifts in focus and the reliance on external context and compare these with the shared objects available through the computer system.

Another way of classifying utterances is by their relation to the task in hand. At one extreme the utterance may have no direct relevance at all, either a digression or purely social. Looking at the task-related conversation, the utterances can be classified into three kinds [335]:

substantive directly relevant to the development of the topic;

annotative points of clarification, elaborations, etc.;

procedural talking about the process of collaboration itself.

In addition, the procedural utterances may be related to the structure of collaboration itself, or may be about the technology supporting the collaboration. The latter is usually in response to a breakdown where the technology has intruded into the communication.

Alison and Brian are now discussing the best way to get to the cinema. Alison is using a whiteboard to draw a map.

1. **Alison**: You go along this road until you get to the river.
2. **Brian**: Do you stop before the river or after you cross it?
3. **Alison**: Before.
4. **Brian**: Draw the river in blue and the roads black . . .
5. **Alison**: So, you turn right beside the river.
6. **Brian**: Past the pub.
7. **Alison**: Yeah . . . Is there another black pen, this one's gone dry?

Alison's first utterance, turn 1, is substantive. Brian then interrupts with an annotative utterance, asking a question of clarification, which is answered by Alison at 3. Brian then makes a procedural point (perhaps prompted by his confusion at 2). In turns 5 and 6, the conversation again becomes substantive, but then the pen runs out, and utterance 7 is a procedural remark concerning the communication technology (pen and whiteboard).

The last form of utterance (procedural technological) is most interesting when analyzing transcripts of computer-mediated conversation as it represents points where the system became apparent to the participants. However, it is also interesting to compare the forms of conversation used in, say, an electronic conference with those in normal speech. For example, a hypertext-oriented conference will allow digressions without any danger of losing the flow of the conference, thus encouraging annotative and procedural utterances.

14.3.4 Breakdown and repair

We have already seen an example of *breakdown* in conversation. When Alison and Brian were talking about Brian's roses, they failed to maintain a shared focus. Brian tried to interpret Alison's utterance in terms of his focus and failed, or rather the meaning in that focus was unusual – greenfly are the symbol of the English summer? He then questioned Alison and the confusion was cleared. This correction after breakdown is called *repair*.

If we look at transcripts of computer-mediated conversations, and see many breakdowns, we should not be surprised: face-to-face and spoken conversations are full of it. We may see breakdowns at many levels. The divergence of topic focus is a quite high level of breakdown. It often becomes apparent when we find failures to identify the referrent of an indexical or deictic reference. Alternatively, such a failure may be due to the speaker using an ambiguous indexical: 'Eustace has just hit Bud . . . he's bleeding' – that is, Eustace is bleeding, Bud was wearing a crash helmet.

At a lower level, we may see breakdown due to incorrectly read gestures or eye-gaze, and through missed or inappropriate back channel responses. For instance, in Section 14.2.5, we described the problems in turn-taking during satellite-based video conferences. It may be difficult to interpret just where a breakdown occurred, as the breakdown may take some time to come to light, and be apparent at a different level from which it began. Alison and Brian are enjoying a day out at a country park:

Alison: Isn't that beautiful?
She points at a stag standing beside a large tree; Brian sees the tree.
Brian: The symmetry of the branches.
Alison: How some people can dislike them I can't understand.
Brian: Yes, the rangers ought to cull those deer, they strip the bark terribly in winter.
Alison: (*Silence*)

The breakdown began with a confused gesture, but led to a divergence of dialog focus. Unfortunately, Brian's remark about the branching (of the tree) could be interpreted in terms of Alison's focus (the stag's antlers) and thus the breakdown did not become apparent until Brian had well and truly put his foot in it. Happily, most breakdowns are detected more quickly, but the deeper the breakdown, and the longer it lasts, the more difficult it is to recover.

Despite the frequency of breakdowns in normal speech, our communication is not usually significantly affected because we are so efficient at repair. (Although Brian may have some difficulty.) Redundancy, frequency of turn-taking and back channels, all contribute to the detection of breakdown and its rapid repair. Electronic communications often reduce redundancy (a single channel), reduce the frequency of turn-taking and reduce back channels. The problem is thus not so much breakdowns in communication, but a reduced ability to recover from them.

14.3.5 Constructing a shared understanding

We have seen that human conversation is in itself inherently ambiguous, relying on context and shared understanding between the parties to disambiguate the utterances. In some spheres, such as legal contracts, the precise meaning out of context becomes very important and thus highly stylized language is used to reduce ambiguity.[3] However, even the legal profession depends on a large body of shared knowledge and understanding about legal terms, case law, etc. Similarly, a book, such as this, attempts to use less ambiguous language and only commonly available knowledge.

The major difference between a book and conversation is that the latter is interactive. The shared knowledge used in a book is static, whereas that used during a conversation is dynamic, as the participants increase their understanding of one another and as they shift their focus from topic to topic.

When participants come to a conversation, they may come from different backgrounds and bring different knowledge. Even close colleagues will have different recent experiences, and as we have seen in previous examples, have different foci. The participants do not try to unify their knowledge and background – indeed, they could not fully do so without living one another's lives. Instead, they seek to obtain a *common ground*, a shared understanding sufficient for the task in hand. Establishing this common ground will involve negotiating the meanings of words and constructing shared interpretations of the world. Clark and Schaefer [72] refer to this process as *grounding*.

A consequence of this model of conversation is that the participants are aware, at various levels both conscious and subconscious, that their common ground is incomplete. Their conversation is not then just an exchange of information about their task, but involves continual testing and cross-checking of the other party's understanding. Consider again a fragment from Alison's conversation about the way to the cinema:

Alison: So, you turn right beside the river.
Brian: Past the pub.
Alison: Yeah . . .

Alison makes an utterance concerning the way to the cinema. Brian interprets this utterance given his current understanding of the conversation and the world. However, in order to check this understanding he makes the statement 'past the pub'. Now this is not a question of clarification like his earlier question about the bridge; instead it merely echoes back some evidence that he has correctly interpreted Alison's utterance. Alison is happy with this and so confirms it 'yeah'.

Such exchanges can be more protracted: for instance, if Brian's reply does not satisfy Alison she may reflect Brian's evidence back to him: 'you mean the Black Bull', or attempt to re-present her original utterance: 'along the road on this side of the river'. Alternatively, the exchange may be much shorter: rather than explicitly presenting evidence, Brian could have simply continued the conversation, making

3 Or left deliberately ambiguous.

his understanding implicit in his future utterances. If these utterances were not compatible with Alison's original utterance, she could then initiate repair. This was the course adopted in their conversation in the country park, although, in that case, with unfortunate results. Finally, the evidence of understanding may often be presented via back channels, little 'yes'es, or simply a continued look of comprehension.

Common ground is always partial, and thus any utterance will have a different meaning for the speaker and the listener. The aim of grounding is to construct a meaning *in the conversation* which is sufficient for the task. For example, Brian's understanding of the Black Bull may be of pleasant evenings sitting on its river terrace. Alison may never have visited the pub, but has seen its distinctive sign hanging over the road. These different understandings are not important: for the purpose of finding directions the pub is merely a way of identifying the road they are to follow.

In a conversation, we know that our partner does not share our knowledge of the world. In addition, we know that our partner will attempt to interpret our utterances. We thus frame our utterances based on this knowledge. Two guiding principles for our utterances are that they should be *relevant* and *helpful*.

To be *relevant* an utterance should further the current topic. This is because our partner is expecting an utterance in this context and any sudden shift in our topic focus will make it more difficult for our partner to make sense of the utterance. Such shifts happen in a conversation, but require less ambiguous utterances (as the common ground for that particular utterance is lower).

To be *helpful*, an utterance should be understandable to the listener and be sufficiently unambiguous given the listener's understanding. This requires the speaker to have a model of the listener's understanding and vice versa. So assuming he is being helpful, in saying 'past the pub', Brian implicitly assumes that there is a particular pub, which Alison will recognize as being significant. It is no good the pub being significant to Brian alone; he must know that it will carry its intended significance to Alison.

The ability to build such models is part of our social maturing. One of the key developmental steps for a child is from an egocentric world view, where things are interpreted in relation to the child, to a social one where the child recognizes others' viewpoints. At the age of 2½, one of the authors' children was interviewed by a linguistics researcher. At one stage the conversation proceeded:

Child:	We went to the doctor.
Researcher:	Where was the doctor?
Child:	Up the steps.

The researcher was clearly (in the context and to an adult) wanting to know whether the doctor was in a hospital or not. The child's answer would have been instantly meaningful to any local parent as the steps to the local doctor were a constant problem for people with prams. However, the child was at that stage unable to phrase the utterance in a way suited to her listener's understanding. At a certain age children assume you know everything they know.[4]

4 To be fair, adults often make the same assumption of children!

So, we see that conversation is an inherently social activity, based on a constructed shared understanding, and relying on the participants' models of one another. In addition, it depends on continuous interaction to correct misinterpretations and to confirm understanding.

14.3.6 Speech act theory

A particular form of conversational analysis, *speech act theory*, has been both influential and controversial in CSCW. Not only is it an analytic technique, but it has been used as the guiding force behind the design of a commercial system, Coordinator. Speech act theory has origins going back over 25 years, but was popularized by Winograd and Flores in the design of Coordinator [381].

The basic premise of speech act theory is that utterances can be characterized by what they *do*. If you say 'I'm hungry', this has a certain *propositional meaning* – that you are feeling hungry. However, depending on who is talking and to whom, this may also carry the meaning 'get me some food' – the intent of the statement is to evoke an action on the part of the hearer. Speech act theory concerns itself with the way utterances interact with the actions of the participants.

Some speech acts actually cause a significant effect by the act of being said. The classic example is when a minister says 'I pronounce you husband and wife'. This is not simply a statement that the minister is making about the couple. The act of saying the words changes the state of the couple. Other acts include promises by the speaker to do something and requests that the hearer do something. These basic acts are called *illocutionary points*.

Individual speech acts can contribute to a conversation. The basic structure of conversations can then be seen as instances of generic conversations. One example of such a generic structure is a *conversation for action* (*CfA*). This is shown as a state diagram in Figure 14.1. It represents the stages two participants go through in

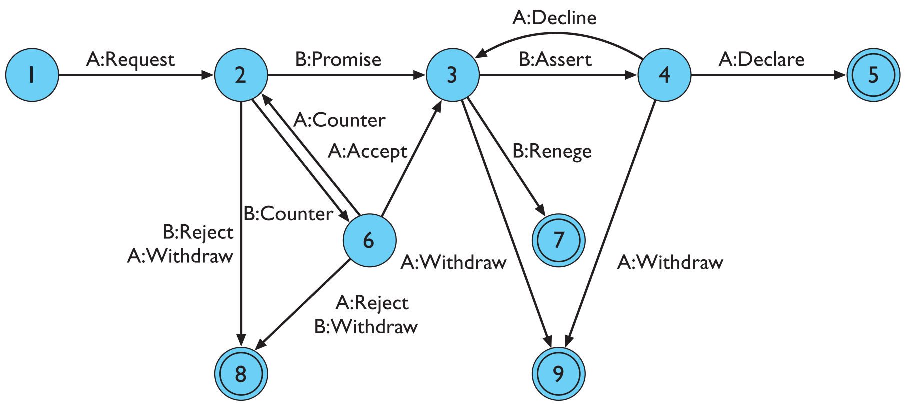


Figure 14.1 Conversation for action. Source: *Understanding Computers and Cognition: A New Foundation for Design* by Terry Winograd/Fernando Flores, © 1986. Reprinted by permission of Pearson Education, Inc., Upper Saddle River, NJ

initiating an external action that one of them should perform. There are two variants, the one shown representing a conversation where the first speaker (A) is requesting that the other participant (B) does something. The other, similar, variant is where the first speaker begins with an offer.

The numbered circles in Figure 14.1 are 'states' of the conversation, and the labeled arcs represent the speech acts, which move the conversation from state to state. Note that the speech acts are named slightly differently in different sources (by the same author even!), but the structure of a CfA is the same. The simplest route through the diagram is through states 1–5.

Alison:	Have you got the market survey on chocolate mousse?
Brian:	Sure.
	Rummages in filing cabinet and hands it to Alison
Brian:	There you are.
Alison:	Thanks.

Alison makes a *request* for the survey (although it is phrased as a question). Brian *promises* to fulfill the request ('sure'). After he feels he has done so (by handing it to Alison), Brian *asserts* that the request has been fulfilled ('there you are') and Alison *declares* her satisfaction that Brian has completed her request ('thanks').

More complex routes may involve some negotiating between the parties. For example, the conversation might have begun:

Alison:	Have you got the market survey on chocolate mousse?
Brian:	I've only got the summary figures.
Alison:	That'll do.

In this Alison's *request* is met by a *counter* from Brian, that is Brian attempts to modify Alison's request. This brings us to state 6 in the diagram. Alison then *accepts* Brian's counter, bringing the conversation back to state 3.

The network has some nodes marked with a double circle. These are the completion nodes, and at these points neither party expects any more acts by the other as part of this conversation. So the fragment above which left Alison and Brian in state 3 must continue. Of these completion nodes only state 5 represents conclusions where the request has been satisfied. For example, Alison's initial request could have been answered with 'it's confidential' (meaning 'you can't have it'). This is the action of Brian *rejecting* Alison's request, leaving the conversation in state 8 and complete.

Not all speech acts need be spoken! Often a silence or an unspoken action forms a speech act. For example, let us imagine that the market survey had not been handy and so Brian answers Alison's request with 'sure, I'll get it later'. Later in the day he finds an electronic copy of the report and then emails it to Alison. His action will be interpreted as *asserting* completion. If Alison does not respond within a short time, her silence will be read as *declaring* satisfaction and the conversation will be completed.

There are other generic conversation forms as well as CfA. These include:

conversation for clarification usually embedded within a CfA to clarify the required action (different from countering a request);

conversation for possibilities looking toward future actions;

conversation for orientation building up a shared understanding.

In addition, the participants may indulge in meta-conversation, discussing the acts themselves, perhaps questioning the legitimacy of an act: 'I'm hungry' . . . 'well I'm not your skivvy, get your own food'. Also CfA is the most extensive and well developed of the conversational forms. For example, the 'creative' conversation for possibilities will have a much less structured form.

The importance of CfA is that actions are central to organizational administration. In the words of Terry Winograd [380]:

> Conversations for action are the central coordinating structure for human organizations.

This belief in the importance of CfA, together with the assumption that making speech acts explicit will aid communication, prompted the design of the tool *Coordinator*. Coordinator is a form of structured email system. When sending a message, the participants must say what kind of illocutionary act they are performing and what part it plays in a conversation. The tool knows about CfA and a few other conversational forms. In addition, it allows time limits to be put on messages. Together these facilities allow the participants to keep track of their own commitments and those of others.

As an example, imagine Alison wishes to use Coordinator to request the market survey from Brian. She selects a menu option saying that she wants to initiate a CfA. The system then offers her two possibilities, a request or an offer. She chooses to make a request and from now on the system knows that a CfA is in progress. She then types 'have you got the market survey on chocolate mousse' into the text area of the message. Note that the system does not try to structure or interpret the natural language *content* of the utterance, but only demands that the sender declares the illocutionary point of it. Brian receives Alison's message and is told by the system that it is a request. He is then offered the various conversational moves that can follow: promise, counter-offer or decline (*reject*), plus a few more not on Figure 14.1. If he chooses promise, the system fills the message area with the default words 'I promise to do as you request', which can be altered (to 'sure') if Brian desires, or sent as it is. The intention is that many simple acts can be completed by the defaults.

Coordinator, being one of the earlier CSCW systems, has had plenty of criticism. Indeed, 'Coordinator bashing' has become so common in CSCW circles that it (the bashing) is coming under criticism itself. There are three main problems: reservations about speech act theory itself and CfA in particular, that people dislike using Coordinator, and whether the whole concept of making intentions explicit is a good idea.

The first criticism is that speech acts do not adequately describe conversations. For example, Alison walks into Brian's office while he is on the phone, he picks a report from the table and gives it to her, she walks out. Speech act theory would regard this as a conversation for action. The request is implicit; presumably Brian knows what Alison wants from some previous context. Then when Brian hands over the report, he is, by complying with her request, both implicitly *promising* to fulfill and *asserting* completion of the request. Finally, by going out of the office Alison implicitly declares completion. There seems to be an awful lot of squeezing to get the interchange to fit the CfA! Indeed, this is recognized in part within Coordinator as a valid response to a request is a report of completion (*assert*), that is taking the *promise* as read.

There have been mixed responses as to the usefulness of Coordinator, but most (certainly the most vocal) have been negative. The conversational forms basically do not do what people want. Those users who continued to use it ended up using 'free-form' messages – a non-interpreted action. Effectively, they used Coordinator as a standard email. It has been claimed that the only organizations that have used Coordinator successfully are those with strong authoritarian managerial structures where the employees have been ordered to use it. Coordinator has even been dubbed 'the world's first fascist computer system'[5] – which was certainly *not* the intention of the designers. As you can see, emotions tend to run high when discussing Coordinator!

The fundamental approach of Coordinator is different from any previous system, and from most since. Rather than starting with technology – build it and play with it – Coordinator started with a theory of communication and then used this to drive design. Such theory-led design is a thoroughly proper design approach. The debatable issue is the way in which the theory was incorporated into the tool.

Coordinator expects its users to make explicit what is normally implicit in our utterances. We hope the reader will have realized by now just how rich human conversation is, and how effective people are at communication. However, one of the fundamental lessons learnt by the expert systems community is that experts *do* things, they do not know *how* they do them. Forcing expert communicators (people) to think about their communication is rather like asking a centipede to think about walking. This all suggests that theory should be used to guide the design, but should not be embedded explicitly within it.

There is a counter-argument, however. First of all, there is some evidence to suggest that teaching managers to recognize their speech acts improves their communication. The extrapolation is that making the acts explicit improves communication, but that is a major extrapolation. A more measured claim would be that explicit representation is *at least* a good tool for training communication skills. The second argument concerns the nature of electronic communication. Although we are all experts at face-to-face communication with all its subtleties, our expertise is sorely challenged when faced with a blank screen. We lack the facilities to make our intentions implicit in our communications and thus explicit means will help.

Whatever the rights and wrongs of Coordinator's design, the evidence is that its users have largely voted with their feet. More recent systems have included a much greater level of user control, allowing users to build conversational structures of their own. Possibly, the structures they build are merely special cases of CfA and other speech act structures, but users clearly prefer to feel that they have the power over the system.

5 Even if one agreed with the sentiment, it would certainly not be the first.

14.4 TEXT-BASED COMMUNICATION

For *asynchronous* groupware (and even some synchronous systems), the major form of direct communication is text based. There are exceptions to this, for instance voice messaging systems and answerphones, and other media may be used in addition to text such as graphics, voice annotation or even video clips. But despite these, text is still the dominant medium.

Text-based communication is familiar to most people, in that they will have written and received letters. However, the style of letter writing and that of face-to-face communication are very different. The text-based communication in groupware systems is acting as a speech substitute, and, thus, there are some problems adapting between the two media.

There are four types of textual communication in current groupware:

discrete – directed message as in email. There is no explicit connection between different messages, except in so far as the text of the message refers to a previous one.

linear – participants' messages are added in (usually temporal) order to the end of a single transcript.

non-linear – when messages are linked to one another in a hypertext fashion.

spatial – where messages are arranged on a two-dimensional surface.

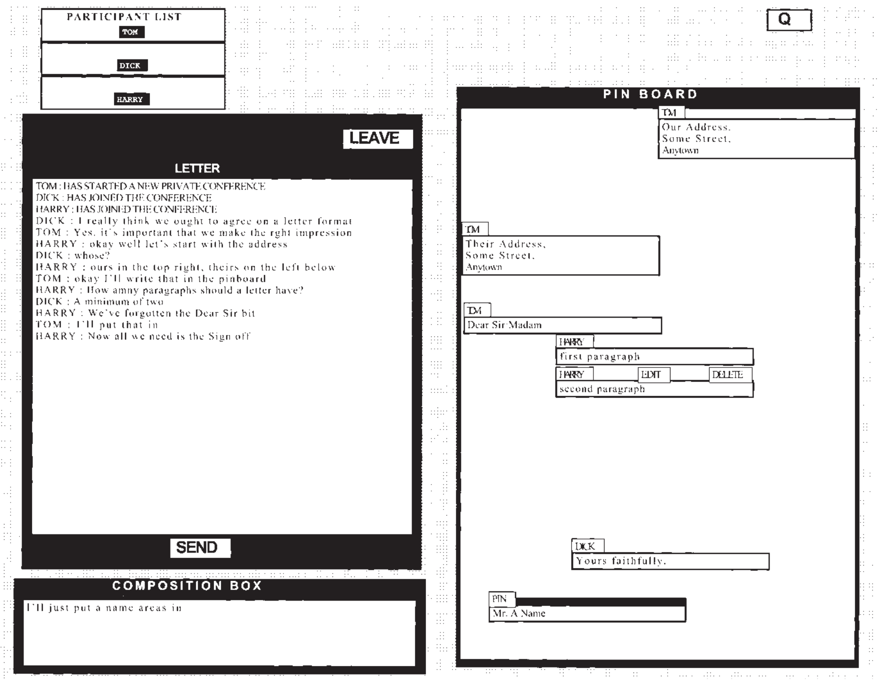
In addition, the communication may be connected to other shared computer artefacts, which will be described further in Chapter 19 (Section 19.6). In the case where the communication is an annotation, the annotation itself may be structured in any of the ways listed above.

A special case of a linear transcript is structured message systems such as Coordinator, where not only the order but also the function of each message is determined. The other extreme is where the transcript is presented as a single stream, with no special fields except the name of the contributor. Figure 14.2 shows a screen shot of the York Conferencer system showing such a transcript on the left of the screen. On the right is an electronic pin-board, an example of spatially organized text.

In this section, we will discuss some of the differences between face-to-face conversation and text-based communications. We will use several of the concepts introduced during our discussion of face-to-face communication and conversational structure in Sections 14.2 and 14.3.

14.4.1 Back channels and affective state

One of the most profound differences between face-to-face and text-based communication is the lack of fine-grained channels. Much of the coordination of face-to-face conversation depends on back channels and interpretation of the listener's expressions. Text-based communication loses these back channels completely. Consider the effect of this on even a two-party conversation. Where the



```
TOM : HAS STARTED A NEW PRIVATE CONFERENCE
DICK : HAS JOINED THE CONFERENCE
HARRY : HAS JOINED THE CONFERENCE
DICK : I really think we ought to agree on a letter format
TOM : Yes, it's important that we make the rght impression
HARRY : okay well let's start with the address
DICK : whose?
HARRY : ours in the top right, theirs on the left below
TOM : okay I'll write that in the pinboard
HARRY : How amny paragraphs should a letter have?
DICK : A minimum of two
HARRY : We've forgotten the Dear Sir bit
TOM : I'll put that in
HARRY : Now all we need is the Sign off
```

```
I'll just put a name areas in
```

```
Our Address,
Some Street,
Anytown

Their Address,
Some Street,
Anytown

Dear Sir Madam

first paragraph

second paragraph

Yours faithfully,

Mr. A Name
```

Figure 14.2 Conferencer screen shot showing text transcript and pin-board

speaker would pause to seek back channel confirmation or to offer the floor, the text 'speaker' must either continue regardless, or finish the message, effectively passing the turn. One consequence of the lack of interruptions and more measured pace of interaction is that the utterances are more grammatical than speech – but still *not* Queen's English!

In addition to this loss of back channels, the speaker's tone of voice and body language are of course absent. These normally convey the *affective state* of the speaker (happy, sad, angry, humorous) and the *illocutionary force* of the message (an important and urgent demand or a deferential request). Email users have developed explicit tokens of their affective state by the use of 'flaming' and 'smilies', using punctuation and acronyms; for example:

:-)	– smiling face, happy
:-(	– sad face, upset or angry
;-)	– winking face, humorous
LOL	– laughing out loud.

People tend to use stronger language in email than in face-to-face conversation, for example they are more likely to be highly and emotively critical. On the other hand, they are less likely to get emotionally charged themselves. These apparently contradictory findings make sense when you take into account the lack of implicit affective communication. The participants have to put this explicitly into their messages – thus accounting for their stronger language. At the same time, they are emotionally 'distanced' by the text from their conversants and have the conversation spread out over time. In addition, they do not have to express their affective state by *acting* emotionally. Together these factors contribute to a more heated conversation by calmer conversants!

The culture of electronic mail

We use or receive many different kinds of paper messages: formal letters, memos, handwritten notes, pre-printed letters and postcards. In addition, we communicate by voice: in meetings and lectures, by telephone and as we pass in the corridor. The same message delivered by different media has a different meaning for us. Someone asks us to do something as we pass them in the corridor, but we forget. We don't feel too guilty. But if the same request came by formal letter we would be expected to respond.

These communication mechanisms have been around a long time and we largely understand the social force of different media. That is, we have a shared culture for paper and voice media. The situation for electronic media is far less clear.

In the past, email used to be very unreliable. So, if someone sent you a mail and you were slow to respond or didn't respond at all, you didn't feel too guilty. After all, for all the sender knew, you might never have received the message! That is, the unreliable nature of the medium built up a culture where the social force of email was weak – rather like the comment while passing in the corridor.

However, increasingly email is used within organizations for passing critical information or making important decisions. The expectation is that email will be treated with the same force as a formal letter or memo.

Imagine what happens when the two cultures meet . . . Sending a message to someone asking them to come to a meeting in half an hour will obviously only have its intended effect if the sender and recipient have a shared culture of use.

Often you cannot know when you send a message what the email culture of the recipient is. It varies between organizations and even between groups and individuals within an organization. The medium itself gives few clues. Even more frightening, many people do not even *realize* that there are such cultural differences.

14.4.2 Grounding constraints

In Section 14.3.5, we discussed the process by which conversants obtain common ground. This grounding process is linked strongly with the types of channels through which the conversants communicate. Clark and Brennan [71] describe the properties of these channels in terms of *grounding constraints*. These include:

cotemporality – an utterance is heard as soon as it is said (or typed);

simultaneity – the participants can send and receive at the same time;

sequence – the utterances are ordered.

These are all constraints which are weaker in text-based compared with face-to-face interaction. For example, simultaneity in face-to-face conversation allows back channel responses. Even where, say, two participants can see each other's typed messages as they are produced, the nature of typing makes it all but impossible to type your message whilst looking for your colleague's 'back channel' response.

In a text-based system, different participants can compose simultaneously, but they lack cotemporality. As we saw, even if the messages appear as they are produced, they will not be read in real time. In addition, the messages may only be delivered when complete and even then may be delayed by slow communications networks.

Linear transcripts obviously have some idea of sequence, but this is confused by the overlap and interleaving caused by the lack of cotemporality and simultaneity. Consider this typical interchange during the use of the York Conferencer system:

1. **Bethan**: How many should be in the group?
2. **Rowena**: Maybe this could be one of the four strongest reasons?
3. **Rowena**: Please clarify what you mean.
4. **Bethan**: I agree.
5. **Rowena**: Hang on.
6. **Rowena**: Bethan what did you mean?

Rowena and Bethan composed their first utterances simultaneously. When Rowena looks up to the transcript area, she sees Bethan's message and does not understand it, so she enters the canned phrase 'Please clarify what you mean' which is generated by a button marked 'Clarify'. Simultaneously, Bethan reads Rowena's message (2) and hits her canned phrase button 'Agree'. Rowena is then confused about what Bethan means by 'I agree' as the preceding message was her request for clarification.

In a spoken conversation, Rowena and Bethan would have quickly corrected themselves if they began to speak at once, and the linearity would have reflected a *common* experience. The trouble is that the participants in the text-based conference each experienced the messages in a different order:

Rowena:	2 1 3 4 5 6
Bethan:	1 2 4 3 5 6

We will discuss these problems of interleaving and overlapped messages further in the following sections.

Altogether, the lack of grounding constraints in text-based communication makes it more difficult to obtain a common ground. It has also been found that email and text-based meetings are less effective at resolving conflicts than a face-to-face meeting.

14.4.3 Turn-taking

We saw that one of the fundamental structures of conversation was *turn-taking* (Section 14.2.5). The last transcript was an example of a breakdown in turn-taking. In fact, such breakdowns are quite rare in two-party electronic conversations and are quickly corrected. What is more surprising is that such breakdowns so rarely occur during letter writing, which is in some ways similar. However, when conversing by letter, one has an objective timescale with which to work out whether one's fellow conversant ought to have replied. One therefore does not send a second letter unless the conversant is very remiss in replying to the first missive. However, in synchronous text-based conversation, the time taken to compose a message (from 30 seconds to several minutes) is far greater than the few seconds which feel 'immediate' on a computer system, but is too short to be able to reason about rationally. The replies always seem a long time coming and hence one is tempted to send a 'follow-on' message.

Despite the occasional breakdown, most observers of two-party text-based interaction report an overall turn-taking protocol, which exhibits many of the structures of normal conversation including *adjacency pairs*. However, when we look at three or more participants, turn-taking and adjacency pair structure begin to break down completely.

In a pair of participants, turn-taking is simple; first one person says something, then the other. The only problem is deciding exactly *when* the exchange should happen. With three or more participants, turn-taking is more complex. They must decide *who* should have the next turn. This is resolved by face-to-face groups in a number of ways. First, the conversation may, for a period, be focused on two of the parties, in which case normal two-party turn-taking holds. Secondly, the speaker may specifically address another participant as the utterance is finished, either implicitly by body position, or explicitly: 'what do you think Alison?' Finally, the next speaker may be left open, but the cotemporality of the audio channel allows the other participants to negotiate the turn. Basically, whoever speaks first, or most strongly, gets in.

These mechanisms are aided by back channels, as one of the listeners may make it clear that she wants to speak. In this case, either the speaker will explicitly pass the turn (the second option above), or at least the other listeners are expecting her to speak. In addition, the movement between effective two-party conversation (the first option) and open discussion will be mediated by back channel messages from the other participants.

In an unstructured text-based conversation the third option is not available, nor, of course, are the back channels. Paired conversation is quite common and the second option, explicitly naming the next speaker, is possible. However, this naming is not particularly natural unless a direct question is being asked. In both options, the absence of back channels makes it difficult for another listener to interrupt the conversation. Some systems use more structured mechanisms to get round these problems, perhaps having a round-robin protocol (each participant 'speaks' in turn) or having a queue of turn-requests. Whether the strictures of such mechanisms are worse than the problems of occasional breakdown depends very much on the context and is a matter of opinion.

14.4.4 Context and deixis

We have seen how important context is in ordinary speech. Utterances are highly ambiguous and are only meaningful with respect to *external context*, the state of the world, and *internal context*, the state of the conversation. Both of these are problems in text-based communication.

The very fact that the participants are not co-present makes it more difficult to use external context to disambiguate utterances. This is why many groupware systems strive so hard to make the participants' views the same; that is, to maintain WYSIWIS ('what you see is what I see'). In Chapter 19 we will look at an example that shows that this is an issue even when the participants have audio/video communications or are in the same room!

Whatever the means of direct communication, remote participants have difficulty in using deictic reference. They cannot simply say 'that one', but must usually describe the referrant: 'the big circle in the corner'. If their displays are not WYSIWIS then they must also ensure that their colleague's display includes the object referred to and that the description is unambiguous. Asynchronous participants have even more problems with deixis as there is no opportunity for their colleagues to clarify a reference (without extremely lengthy exchanges). Furthermore, the objects referred to by a message may have changed by the time someone comes to read it! Similarly, group pointers are not really an option, but one can use methods of linking the conversation to its context, either by embedding it within the objects as annotations or by having hypertext links between the conversation and the object.

The trouble does not end with external context; there are also problems with deictic reference to internal context. In speech, the context is intimately connected to linear sequence and adjacency. As we have seen, even in linear text transcripts, overlap breaks the strict sequentiality of the conversation, and thus causes problems with indexicals and with context in general.

1. **Alison**: Brian's got some lovely roses.
2. **Brian**: I'm afraid they're covered in greenfly.
3. **Clarise**: I've seen them, they're beautiful.

Brian and Clarise both reply to Alison's message at the same time. However, in the transcript, where Clarise says 'they' this appears, at first, to refer to the greenfly. Brian is expecting a consoling reply like 'I've seen them. Have you tried companion planting?' Of course, the breakdown quickly becomes apparent in this case. The problem is not so much that people cannot recover from such breakdowns, as in the extra burden the recovery puts on the participants. If these messages are being sent, say, between continents, network delays and time differences may limit exchanges to once a day. Even one or two messages recovering from breakdown are then a major disaster.

Most email systems and some bulletin boards lack any implied sequentiality and thus any context to the messages. The users (ever inventive) get round this by including copies of previous messages in their replies. This is only partially effective and, of course, incredibly clumsy.

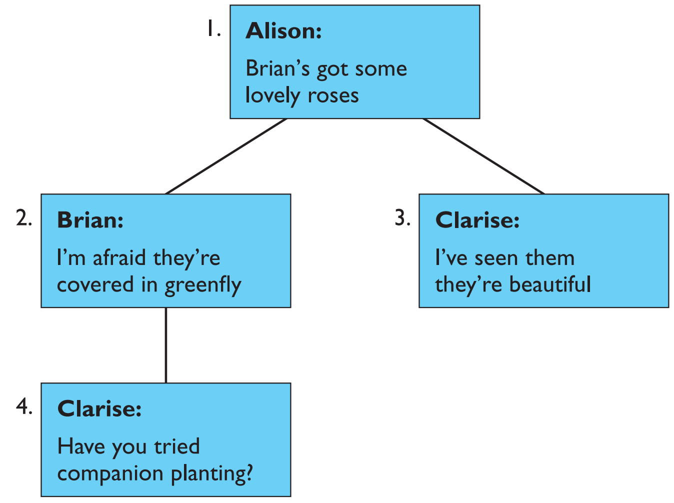

1. Alison: Brian's got some lovely roses
2. Brian: I'm afraid they're covered in greenfly
3. Clarise: I've seen them they're beautiful
4. Clarise: Have you tried companion planting?

Figure 14.3 Hypertext conversation structure

Hypertext-based systems avoid the implied sequentiality of a linear transcript. In the above example, both Brian and Clarise replied to Alison's message at the same time. In a hypertext these would form parallel conversations. This is shown in Figure 14.3, where in addition Clarise has sent a second message offering advice on Brian's greenfly. The use of 'they' in Clarise's message (3) is now perfectly clear.

14.4.5 Pace and granularity

In a spoken conversation, the turns are often only a few tens of seconds long. If we take into account minor confirmations and back channels, the pace is still faster, perhaps a turn or back channel response every few seconds. Compared with this, the pace of email is very slow: messages can take from a few seconds to hours to deliver. Even synchronous text-based conversations are limited by the participants' typing speed and have a pace of at most one turn every minute or so.

The term *pace* is being used in a precise sense above. Imagine a message being composed and sent, the recipient reading (or hearing) the message and then composing and sending a reply. The pace of the conversation is the rate of such a sequence of connected messages and replies. Clearly, as the pace of a conversation reduces, there is a tendency for the *granularity* to increase. To get the same information across, you must send more per message. However, it is not as easy as that. We have seen the importance of feedback from listener to speaker in clarifying meaning and negotiating common ground. Even most monologs are interactive in the sense that the speaker is constantly looking for cues of comprehension in the listener. Reducing the pace of a conversation reduces its *interactivity*.

As well as at the small scale of clarifying individual utterances, interactivity is important in determining the direction of a conversation. Imagine that the conversation

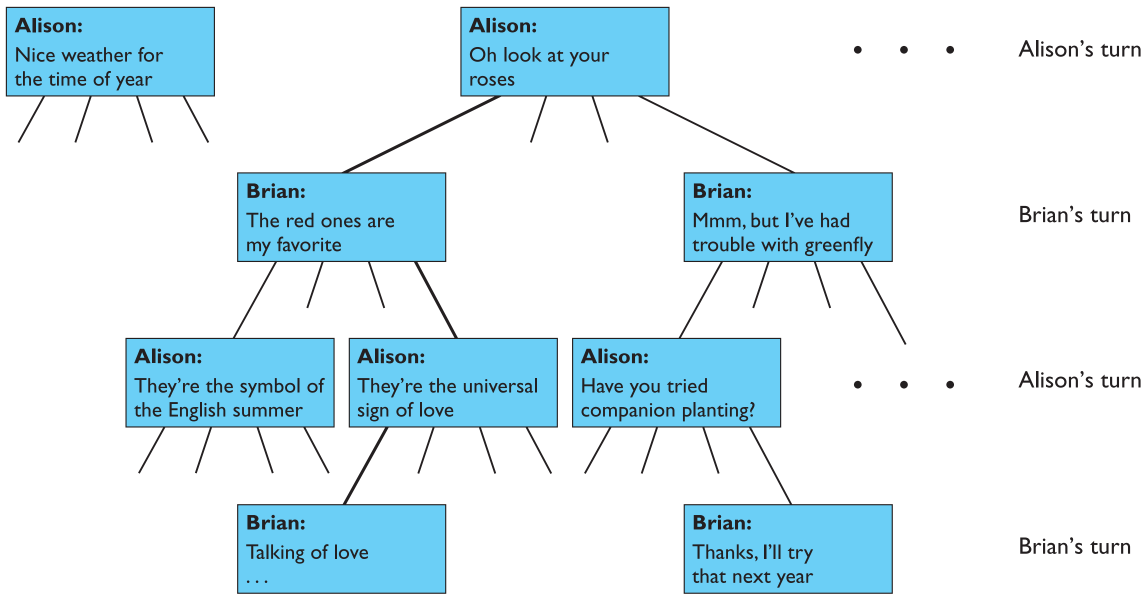

Alison:
Nice weather for the time of year
Alison:
Oh look at your roses
• • •
Alison's turn
Brian:
The red ones are my favorite
Brian:
Mmm, but I've had trouble with greenfly
Brian's turn
Alison:
They're the symbol of the English summer
Alison:
They're the universal sign of love
Alison:
Have you tried companion planting?
• • •
Alison's turn
Brian:
Talking of love . . .
Brian:
Thanks, I'll try that next year
Brian's turn

Figure 14.4 The conversation 'game'

is a little like a game, where the participants can make moves. In Figure 14.4, we can see some of the moves Alison and Brian can make whilst talking in the garden (Clarise has gone home). At each turn of the conversation, Alison or Brian can choose to say one thing which continues the discussion. That is, they gradually work out a path from the top of the tree downwards. A particularly promising conversation path is shown with bold lines.

In a hypertext-based system one can expand several branches of a conversation tree, but in speech or in a linear text transcript the conversation follows one branch. Whatever medium is used, you cannot normally progress down the tree faster than the pace of the conversation. To overcome these limitations, people adopt several *coping strategies*.

The simplest strategy is just to avoid conversation. This can be done by delegating parts of a task to the different participants. Each participant can then perform much of the task without communication. They must still communicate for large-scale strategic decisions, but have significantly reduced the normal communications. Of course, this approach reduces *communication* by reducing *collaboration*.

More interesting in a cooperative work setting are two coping strategies which increase the chunk size of messages in order to reduce the number of interactions required to complete a task. These strategies are frequently seen in both text-based conferences and in letter writing.

The first of these coping strategies is *multiplexing*. Basically, the conversants hold several conversations in parallel, each message referring to several topics. In terms of the conversation tree, this corresponds to going down several branches at once. For an example, consider the transcript in Figure 14.5 taken during a computer-mediated

E: I don't like the other three being in cahoots – shall I form an alliance with one to stitch them up? I'll move to Holland first.

Figure 14.5 Excerpt of transcript from Hewitt et al. [171]. Source: Courtesy of Professor Nigel Gilbert

game of Diplomacy as part of the TMPI (theories of multi-party interaction) project [171]. The turn by England (E) introduces two topics: the forming of an alliance (for subversive purposes), and a particular line of attack (through Holland).

The second coping strategy for increasing the size of message chunks is *eagerness*. The participants can foresee the possible course of the interaction and frame communications which encompass many of the possibilities: for example, 'If you don't pay within seven days we will take you to court'. Thinking of the conversation tree, eagerness is a sort of depth-first strategy. The participant explores a branch of the tree guessing the other participant's responses.

If we compare spoken with written communication we find that letters are far more eager than speech. When writing a letter, one takes more care that the points are stated clearly, and one may even consider alternative responses of the recipient and state one's position on each. For instance, a letter may say 'if you marry me I will be happy for ever, but if not, life will lie like bitter herbs upon my tongue' (lovers have a tradition of being overeager). In extremis one frames a communication which describes one's reaction in all possible situations.

One can find similar incidents of eagerness in electronic conversations, for example the messages in Figure 14.6 from Severinson Eklundh's corpus [317], quoted by Bowers and Churcher [42], both exhibit eagerness, the contingent part of each message being introduced by the key phrase 'In that case'.

Eagerness is less likely to lead to breakdown, except where the message tries to foresee too great a breadth of possibilities and becomes confusing. However, there are various circumstances: for instance, in many process control tasks, where the number of possibilities at each stage of the tree is large. In this case eagerness cannot solve the communication problems.

A: Subject: Report C 123660
The above mentioned report is out of stock.
The remaining ones are C 12366 + C 123660.
What to do? Reprint? In that case, do you have any changes to suggest?

. . . . . .

A: Subject: SIGSIM meeting
Are you going to Linkoping tomorrow?
In that case when are you leaving?
Does SIGSIM pay for the trip or what?

Figure 14.6 Excerpt from Severinson Eklundh's corpus [317]. Source: Courtesy of Kerstin Severinson Eklundh

A potential problem of eagerness is that by following a particular branch of the conversation, other branches, which your colleague would have liked to explore, are missed. In spoken conversations it is quite difficult to return to a previous point. It is possible to say 'going back to . . .', but this can form a potentially rude break in the conversation. In text-based communications, the *reviewability* of the medium reduces this effect. It is easier to return to a missed point as both participants can refer to the conversation up to that point. In addition, the break in the line of conversation is less rude as both participants know that the current topic can itself be picked up again. Finally, there is even the option of multiplexing the current topic with the lost point.

Reviewability is another grounding constraint of communication, but this time one where text-based communication has the advantage over speech. You can of course tape speech, but it is far from easy to use this as a review mechanism.

14.4.6 Linear text vs. hypertext

Considerations of potential overlap suggest that hypertext-based communications may be better suited as a text-based communication medium. Similarly, the problems of pace may be partially solved in a hypertext. Multiplexed messages can be represented as updates to several parts of the hypertext, thus reducing the likelihood of breakdown and lost topics. In addition, if the messages themselves can be mini-hypertexts, then eager messages listing several possible courses of action can be explicitly represented by the message.

On the other hand, hypertext has its disadvantages. Even static hypertexts, which have been carefully crafted by their authors, can be difficult to navigate. A hypertext that is created 'on the fly' is unlikely to be comprehensible to any but those involved in its creation. Conklin and Begeman, themselves associated with the hypertext-based argumentation tool gIBIS, conclude that 'traditional linear text provides a continuous, unwinding thread of context as ideas are proposed and discussed' [76]. For the asynchronous reader trying to catch up with a conversation, a linear transcript is clearly easier, but it is precisely in more asynchronous settings where overlap in linear text is most likely to cause confusion.

We can see that there is no best solution, with possibly the best course in many situations being linear transcripts arranged by topic, with some automatically generated indication of overlap.

14.5 GROUP WORKING

So far we have been principally looking at the properties of direct communication, and largely two-party conversations. Group behavior is more complex still as we have to take into account the dynamic social relationships during group working. We will begin by looking at several factors which affect group working, and then discuss the problems of studying group working. This section deals with groups that are actively working together, rather than the organizational issues considered in the previous chapter, which are primarily concerned with the long-term structures within which people work.

14.5.1 Group dynamics

Whereas organizational relationships such as supervisor/supervisee are relatively stable, the roles and relationships within a group may change dramatically within the lifetime of a task and even within a single work session. For example, studies of joint authoring have found that roles such as author, co-author and commentator change throughout the lifetime of a document [254, 295]. This means that systems, such as co-authoring systems, which use a formal concept of *role*, must allow these roles to change together with the socially defined roles.

Even the naming of roles can cause problems. A person may be an author of a book or paper, but never write the words in it, acting instead as a source of ideas and comments. A particular case of this is the biographical story where the individual concerned and a professional writer co-author the book, but only the professional author writes. A co-authoring system such as Quilt would call the non-writing author a 'commentator' or a 'reviewer', but *not* an 'author'. One can imagine some of the social friction such naming will cause.

Within the microcosm of group interaction, authority roles can be entirely inverted. For example, if the managing director of a coal mining company visits the coal face, he should act under the authority of the supervisor at the face, for his own safety and that of the mine. These inversions can cause problems even in computer-free situations – it is hard for the supervisor to say 'No' to the MD. But, if a system demands an explicit controlling role, it is even harder for the manager to relinquish this explicit role, even if in the context the subordinate should be in control.

Not only do the social relationships *within* the group change, but the group membership and structure can change in time. A member leaving or a new member joining can cause dramatic changes in the behavior of the group. For example, if a very dominant member leaves, the group may change from a leader–follower to a democratic structure. New members have special problems adapting to the particular group subculture, which can develop very quickly among close colleagues. In addition to this social adaptation, the new member must 'catch up' with the substantive work of the group. Groupware systems, for example *argumentation tools*, can help in that they record the history of the group. Groupware designers should in general be aware that new members can and will enter the group and should design their software accordingly. For example, a latecomer to a synchronous conference should be able to review all past contributions, not just the new ones.

The group may also divide into subgroups for detailed discussion and then reform. Tools must be able to support this. For example, early versions of *CoLab's* software only catered for a single WYSIWIS screen – that is, they only supported a single group. Later versions were forced to allow subgroups to work independently and then share results. Note that the CoLab meeting room only has room for six persons; in larger meeting rooms subgroup working is the norm.

14.5.2 Physical layout

In Section 14.2, we discussed the importance of eyegaze and gesture in face-to-face communication and how these help to mediate turn-taking. In particular, we noted in Section 14.2.3 that we must ensure that monitors do not block the participants' views of one another. In general, the physical layout of a room has a profound effect upon the working relationship of those in it. This is particularly obvious for meeting rooms, but should be considered in any group-working environment.

As well as being unobtrusive, the orientation of computing equipment can affect group working. If we wish to encourage conversation, as we do in a meeting room, the participants must be encouraged to look toward one another. Meeting rooms have a natural focus toward the screen at the front of the room, but inward-facing terminals can counteract this focus and thus encourage eye contact [226].

The designers of Capture Lab, an eight-person meeting room, considered all these features and many other subtle effects. However, the users still had some difficulty in adapting to the *power positions* in the electronic meeting room. At first sight, the electronic meeting room is not unlike a normal conference room. If the shared screen is a whiteboard or an overhead projector, then the most powerful position is toward the front of the room (seats 1 or 6 in Figure 14.7). Managers would normally take this seat as they can then easily move to the whiteboard or overhead projector to point out some item and draw the group's attention.

Unless primed beforehand, managers of groups using Capture Lab took one of these seats, but quickly became uncomfortable and moved. In the electronic meeting room, there is no advantage to being at the front, because the screen can be controlled from any terminal. Instead, the power seat is at the back of the room (seats 3 or 4), as from here the manager can observe other people whilst still seeing the screen. Also, the other participants have to turn round when the manager speaks, adding to the manager's authority over the meeting.

14.5.3 Distributed cognition

In Chapter 1, we discussed human cognition, but the emphasis was, as in all traditional psychology, upon the activity *within* the person's head. A school of thinking has recently developed which regards thinking as happening not just within the head, but in the external relationships with things in the world and with other people. This viewpoint is called *distributed cognition* [208, 185].

In fact, this viewpoint is not as radical as it first appears. Traditional views talk about the movement of information between *working memory* and *long-term memory*: it is not so difficult then to regard bits of paper, books and computer systems as extensions to these internal memory systems. Similarly, many models of human cognition regard the mind as a set of interacting subsystems (see Chapter 12): the step to regarding several people as involved in joint thinking is not difficult.

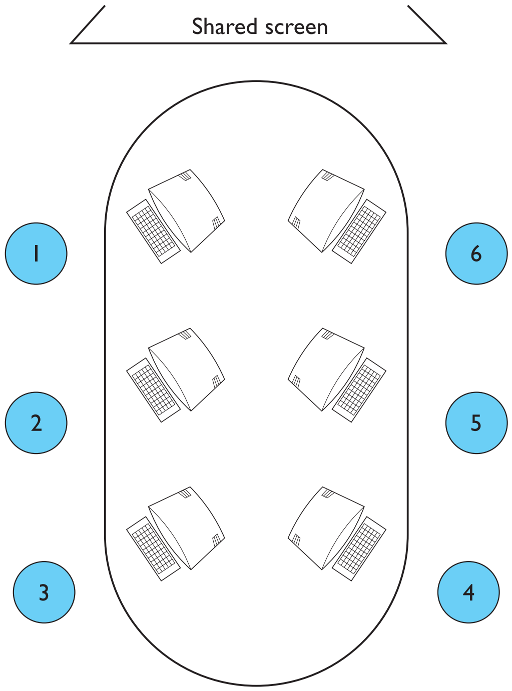


Figure 14.7 Meeting room layout

Remember that this is a view of cognition, that is thinking; it says nothing about awareness, personality or individuality.

Distributed cognition has profound effects on the way we look at group working and even individual work. It emphasizes the importance of *mediating representations*, for example the drawings on a whiteboard. These are no longer just a means of communicating between the parties, but can be a concrete embodiment of group knowledge. Furthermore, it constantly reminds us that communication is not just about getting knowledge from one person's head to another, but about the creation of new group knowledge, not necessarily grasped in totality by any single member.

The emphasis on external forms is encouraging for a designer. It is not necessary to understand completely the individual's cognitive processing in order to design effective groupware. That is an impossible task. Instead, we can focus our analysis of existing group situations and design of groupware on the external representations used by the participants.

14.6 SUMMARY

Face-to-face communication is extremely complex. People maintain precise distances, which can be disrupted through video links. Each utterance is mediated by subtle back channel responses signifying agreement, or on attempting to interrupt.

Body movement, facial expression, eye contact and eyegaze are all used for these back channels and also to establish context.

At a higher level, the structure of conversation can be seen as a sequence of turns, usually alternating between the participants. Context is important in disambiguating utterances, especially when deictic reference is also used. This also depends on the participants establishing a common understanding during the conversation. Breakdowns do occur in conversation, but conversational repair is very effective. Speech act theory, a detailed analysis of conversational structure, has been used to drive the design of Coordinator, a highly controversial, commercial, structured messaging system.

Text-based communication loses most of the low-level feedback of face-to-face conversation. This, and the possibility of overlapping turns, makes it more difficult to establish the context of a textual utterance, and therefore to disambiguate deixis. The reduced pace of text-based conversation means that participants are forced to increase the granularity of their messages. They may achieve this by multiplexing messages or by being eager, predicting their colleagues' responses.

Group dynamics make it very difficult to predict how a particular group will behave. In particular, small things, such as the layout of chairs in a room, can have a major effect. We can see the thinking in a group as being distributed, not locked in any individual, but being within the whole group and the physical representations they use.

Exercises

14.1 In Section 14.3.2, we discussed the highly contextual nature of the spoken word, including the use of deictic reference and indexicals, and the (officially) ungrammatical and fragmentary use of sentences. Try listening to social chat over cups of tea – collect examples of different forms of contextual utterance.

14.2 Go into an office or other place where several people are working together. Try to note down in as much detail as possible what they are doing and when. Do this with different foci: focus on the direct interpersonal communications, focus on the shared objects such as a calendar or document, or focus on one worker at a time. Whilst collecting data and when ordering your notes, look for breakdowns and misunderstandings, and for implicit communication through objects. Look also at a particular task over a period of time, and note the number of interruptions as a worker performs the task, or the way a single task is contributed to by several workers.

14.3 What is speech act theory? Describe positive and negative issues that have arisen when it has been embodied in a specific system.

14.4 Compare turn-taking, round-robin and free-for-all as floor control mechanisms. When might each be effective?

RECOMMENDED READING

J. Carroll, editor, *HCI Models, Theories, and Frameworks: Toward an Interdisciplinary Science*, Morgan Kaufmann, 2003.
See chapters by Perry on distributed cognition, Monk on common ground and Kraut on social psychology.

L. A. Suchman, *Plans and Situated Actions: The Problem of Human–Machine Communication*, Cambridge University Press, 1987.
This book popularized ethnography within HCI. It puts forward the viewpoint that most actions are not pre-planned, but situated within the context in which they occur. The principal domain of the book is the design of help for a photocopier. This is itself a single-user task, but the methodology applied is based on both ethnographic and conversational analysis. The book includes several chapters discussing the contextual nature of language and analysis of conversation transcripts.

T. Winograd and F. Flores, *Understanding Computers and Cognition: A New Foundation for Design*, Addison-Wesley, 1986.
Like Suchman, this book emphasizes the contextual nature of language and the weakness of traditional artificial intelligence research. It includes an account of speech act theory as applied to Coordinator. Many people disagree with the authors' use of speech act theory, but, whether by application or reaction, this work has been highly influential.

S. Greenberg, editor, *Computer-supported Cooperative Work and Groupware*, Academic Press, 1991.
The contents of this collection originally made up two special issues of the *International Journal of Man–Machine Studies*. In addition, the book contains Greenberg's extensive annotated bibliography of CSCW, a major entry point for any research into the field. Updated versions of the bibliography can be obtained from the Department of Computer Science, University of Calgary, Calgary, Alberta, Canada.

Communications of the ACM, Vol. 34, No. 12, special issue on 'collaborative computing', December 1991.

Several issues of the journal *Interacting with Computers* from late 1992 through early 1993 have a special emphasis on CSCW.

Computer-Supported Cooperative Work is a journal dedicated to CSCW. See also back issues of the journal *Collaborative Computing*. This ran independently for a while, but has now merged with *Computer-Supported Cooperative Work*.

See also the recommended reading list for Chapter 19, especially the conference proceedings.

21 Hypertext, multimedia and the world wide web

OVERVIEW

- Hypertext allows documents to be linked in a non-linear fashion.
- Multimedia incorporates different media: sound, images, video.
- The world wide web is a global hypermedia system.
- Animation and video can show information that is difficult to convey statically.
- Applications of hypermedia include online help, education and e-commerce.
- Design for the world wide web illustrates general hypermedia design, but also has its own special problems.
- Dynamic web content can be used for simple online demonstration or for complete web-based business applications.

21.1 INTRODUCTION

Increases in desktop computing power have enabled the rapid growth of the multimedia industry. CD-ROMs, stereo sound and often video input/output are now part of the specification of standard personal computers. Furthermore, the dreams of Vannevar Bush and Ted Nelson (see Chapter 4) have now become reality in the world wide web, which links computers, information and ultimately people across the world.

In this chapter we look at the distinctive features of hypertext and multimedia, where they are used and their potential problems. The world wide web is just one example, albeit an important one, of a hypertext system. We will first consider some of the common features of hypermedia including different types of content and application areas. Diagrams and photographs can enliven text, but some of the most exciting effects are seen when the computational power of the computer is used to allow animation, video and interactive features. We will also consider in Section 21.3 the 'lost in hyperspace' problem, how to design hypertext that is easy to understand and to navigate and how features such as bookmarks, the back button and search can make hypertext and the web accessible. The chapter then focusses on the web. We will look at the technology that underlies the web and how issues of networking influence usability. We will then consider static web content including graphics and streamed media. Finally, we will review ways in which web pages can be made dynamic through interactive content and through server-side generation of web pages from databases.

21.2 UNDERSTANDING HYPERTEXT

21.2.1 Hypertext definition – text, hypertext and multimedia

Although pictures and sculpture came first, it is text – the written word – that is commonly seen as the defining point of civilization; pre-literate and oral societies are often regarded as pre-history simply because they have no extant story. Hieroglyphics, Babylonian mud tablets, the Book of Kells, the Caxton press: these are the stepping-stones toward our current information-centric society.

All these traditional texts share a common linear nature. Aristotle in his *Poetics* said that a story must have a beginning, a middle and an end, and even postmodern non-linear narrative is actually written in a linear fashion even though the events may not be causally connected.

This linearity is partly because of the nature of the media used – papyrus scroll, painted frieze or paper book – but perhaps more significantly because we are creatures in linear time. We are natural story-tellers and natural story-hearers. This is why, in Chapter 5, we found that scenarios were so powerful. The skill of the author is in producing an experience for the reader that introduces new events and new concepts so that they fit meaningfully into what has gone before.

However, there are classes of activities where the reader needs to establish their own path through a text. For example, in the online documentation or help systems discussed in Chapter 11, the user wants to know the right thing for him at that moment, not a full description of the system (although of course tutorials are typically linear). During some forms of exploratory learning the learners may want to follow their own paths through material: each one delving into details in different parts. Experts in a subject, too, may well want to remind themselves of some particular issue or fact.

For such purposes the linear form of a traditional text is a hindrance. For example, when using manuals, the user may not understand all the terms used in the text, and will have to keep going back to a different series of pages to look up the definitions, returning to the original pages and trying to pick up the thread of discussion afterwards. Paper books include tables of contents or indexes and encyclopedias and dictionaries are designed for non-linear reading. However, still the reader is left physically skipping back and forth.

Hypertext attempts to get around these limitations of text by structuring it into a mesh rather than a line. This allows a number of different pages to be accessed from the current one, and, if the hypertext is well designed, the user should find it easier to follow his own particular idea through the mesh rather than being forced down one route. Typically, hypertext systems incorporate diagrams, photographs and other media as well as text. Such systems are often known as *multimedia* or *hypermedia* systems, although the three terms are often used interchangeably.

A hypertext system comprises a number of pages and a set of *links* that are used to connect pages together. The links can join any page to any other page, and there can be more than one link per page. Thus a hypertext document does not simply start a linear progression and follow it to an end, but goes in lots of different directions, some of which terminate, while others link back into different parts of the document (see Figure 21.1, which illustrates the difference between linear text and hypertext).

There are many different ways of traversing the network, and so there are many different ways of reading a hypertext document – the intention is that the user is able

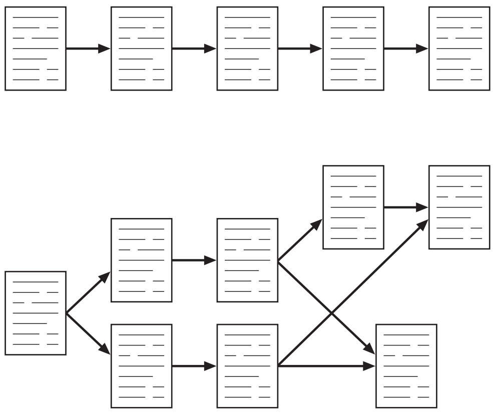


Figure 21.1 Typical structures of linear text and hypertext

to read it in the way that suits him best. Links can exist at the end of pages, with the user choosing which one to follow, or can be embedded within the document itself. For example, in an online manual, all the technical words may be linked directly to their definitions in the glossary. Simply clicking on an unknown word takes the user to the relevant place in the glossary. Another unknown word encountered there can also be traced back to its definition and then the user can easily return to his original place in the manual. The positions of these links are known as *hot-spots* since they respond to mouse clicks. Hot-spots can also be embedded within diagrams, pictures or maps, allowing the user to focus his attention on aspects that interest him.

21.2.2 Rich content

As well as static material – text and static diagrams and photographs – hypertext systems may also include more dynamic material such as animation, video and audio clips, and even full computer applications.

Animation

Animation is the term given to the addition of motion to images, making them move, alter and change in time. A simple example of animation in an interface is in the form of a clock. Digital clocks can flick by the seconds, whilst others imitate Salvador Dali and bend and warp one numeral into the next. Analog clocks have moving hour and minute hands, with an optional second hand sweeping round the clock face. Such a desktop accessory is found in a lot of interface setups, and the additional processing time required to produce such effects is no longer a major factor.

Another common use for animation in current windowing systems is to animate the cursor. Instead of simply having a basic pointer always on the screen, many interfaces now use the typical 16×16 bitmap that makes up the cursor to indicate more complex information. We saw in Chapter 3 that there are a number of different static cursors in use, but animation takes this one stage further by adding motion to the images. This is usually done to indicate that some process is in progress, to confirm to the user that something is actually happening. Animating the cursor means that messages do not need to be printed out to a window, making it a neat and concise way of presenting the desired information. On the Macintosh, work in progress is indicated by a watch icon, with the hands moving round and round, or by a spinning disk. One system uses a stick person, apparently doing weightlifting, to show that heavy work is in progress, whilst another has an hourglass trickling down.

Non-cursor process indicators often take the form of a pop-up box with a moving slider, or a stick person walking backwards and forwards. These are important to give the user a sense of progress and to prevent inactivity being interpreted as an indication of a system fault. In addition, *animated icons* may be used to show system state; for example, Netscape Navigator has shooting stars to indicate that it is downloading information. Animated icons can also be used to make the meaning of the icon clearer. This is especially useful for icons representing actions that are not well represented by static images. However, such animation must be used sparingly to avoid distraction!

Animation can be used to great effect to show changes in data sets, where slow fluctuations can be visualized with the help of rippling three-dimensional colored surfaces, or abrupt changes shown by sudden discontinuities in an otherwise regular motion.

Animation is also used in a cartoon-like way, where animated objects are used to perform particular functions for the user. For example, in an animated help system, a character can appear on the screen and interact with the user, guiding him through a number of stages of help before being dismissed. Such a guide can be endowed with a certain character, making the interaction less impersonal and more interesting for the user, who becomes more involved with the system and is happy to learn more about it.

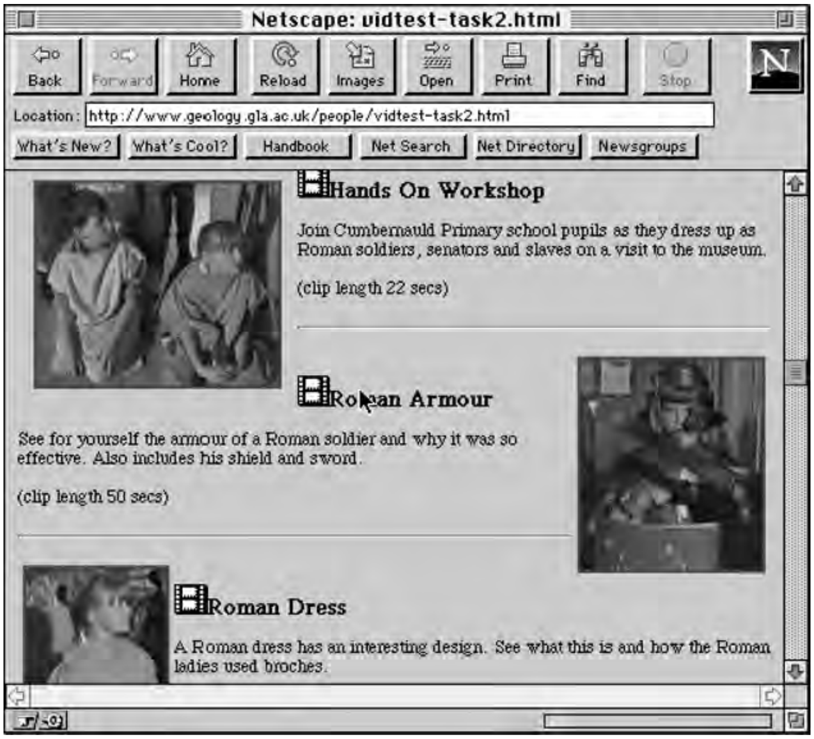
Talking heads

One use of animation is to produce *talking heads* [359]. These are images of human faces either completely computer generated or derived from photographs of real people. They are based on models of human anatomy so that realistic lip movement can be synchronized with generated speech. Surprisingly, this is not just a fun activity but is actually useful! One important use is for deaf or partially deaf users who can lip read the image. Another application is to synchronize the lips with telephone speech allowing a form of video phone over low-bandwidth telephone lines.

These faces can also express emotion. Users have imputed personality to computers since they only had flashing lights. By simulating human expressions the designer can express intended affective states rather than leaving this to the user. For example, this could be useful in a learning environment where the students might naturally view the computer as a harsh, unforgiving teacher, but by means of a suitable face and expressions could be induced to regard it as a colleague or facilitator in learning.

Video and audio

In a media dominated world, there are strong arguments for using video or audio material as part of hypertext systems whether for education, entertainment or reference. Both audio and video material are expensive and time consuming to produce, but increasingly even home-PC systems include video and audio editing as standard. For example, the iMac includes a suite for editing video and burning DVDs. Combined with digital video cameras these bring the production of audio/video material into the reach of many. Furthermore, standard formats such as QuickTime allow this material to be embedded in web pages for easy distribution. Of course, quality video production requires extensive experience, but then so does text!



Hands On Workshop

Join Cumbernauld Primary school pupils as they dress up as Roman soldiers, senators and slaves on a visit to the museum.

(clip length 22 secs)

Roman Armour

See for yourself the armour of a Roman soldier and why it was so effective. Also includes his shield and sword.

(clip length 50 secs)

Roman Dress

A Roman dress has an interesting design. See what this is and how the Roman ladies used broches.

Figure 21.2 Hunterian Museum – learning about the Romans. Source [194a] reprinted with permission of Springer-Verlag; Netscape browser window © 2002 Netscape Communications Corporation, used by permission

Perhaps the biggest problem with audio and video is still the memory requirements. As we saw in Chapter 2, this is becoming more manageable as memory capacities increase, but it will probably continue to be a major problem, especially for web-based material, for some time to come.

The most common use of video is as a clip to be embedded within a text or graphical document or web page. For example, Figure 21.2 shows an example of a prototype website produced by Glasgow University for the Hunterian Museum [194] including video clips. (Notice, too, the use of the still picture for each video clip and the indication of the length of the clip.)

Longer video sequences are, of course, more linear than plain text. This may be acceptable if the hypertext is acting effectively as an index for video material. For example, one might have a collection of silent movies online and access them through a website. However, if the video is regarded as an integrated part of the hypermedia experience it needs to be active in some way as well. It is possible to have hotspots in video images just as in graphics; however, clicking on a gymnastics star to get information on her recent performance may be difficult if she is in the middle of an exercise on the asymmetric bars. Interactive television manages this by having a smaller number of context sensitive options, or images may be frozen to enable exploration.

Perhaps gaming stations best manage this interaction between video and other material, as clips of live action or pre-recorded high-quality generated images are interspersed seamlessly with game playing.

Audio material can be used in the same way as ‘clips’ within a document; for example, in an interactive guide to Beethoven’s music. Because most hypertext material is visual, audio can also be used for background music or sound effects to establish mood, or to provide ‘voice-over’ for static graphics or animations.

Audio material may be stored as sound samples, that is digital recordings, which can include instruments, voice or other sounds. As we are often more sensitive to poor sound than poor video images, and sounds are, by definition, constantly changing, it can be harder to *compress* sound than video. MP3, used for much web downloadable music, recruits knowledge of the auditory system to achieve its high compression. For example, if there are two simultaneous sounds, one a lot louder than the other, the quieter one can be played with less fidelity because its details are masked by the louder sound. For pure instrumental music, MIDI files simply record which instrument must play and the duration, pitch and loudness of each note – basically a digital form of the sheet music.

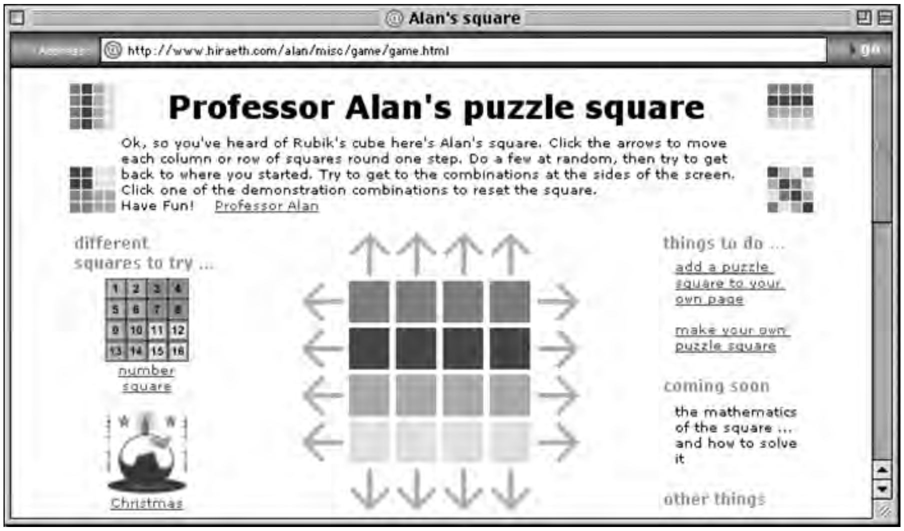
Any sound in the interface can, of course, be potentially annoying, especially in open-plan working environments, and so audio should be used with care and it should be easy to mute. As these words are being written, the current author’s wife is struggling to turn off an annoying musical backing to a web page she is viewing!

Moving pictures are excellent at conveying information, and exert a quite hypnotic hold over us; note the success of television. It is fair to say that many design practices will have to be updated to make the most of the possibilities that these techniques offer; moreover, it is also the case that people are unsure how to get the most out of such technology. The techniques required to gain maximum benefit from moving images are very different from those that are used for static or minimal motion displays, and designers do not have enough experience to start applying the relevant technology at the relevant time. It may well be that computer interface designers will have to study the techniques of the film makers and cartoonists before they start to discover the real benefits that these techniques can provide. Indeed, partly due to Brenda Laurel’s influential book *Computers as Theatre* [207], drama theory is now a respectable and popular topic in HCI.

Computation, intelligence and interaction

Of course the good thing about a computer is that not only can it *show* things that have previously been prepared, it can also *do* things. This book has an index, but it does not contain every word in the text (it may also refer to parts of the text that do not have a particular word). However, the web search can look through all the chapters and find any words you want.

More interactive hypermedia may contain embedded games or applications. For example, Figure 21.3 shows a puzzle from the website of one of the authors (Alan), a sort of 2D Rubik’s cube that you can play online. Hypermedia running on the user’s own computer may interact closely with other applications; for example, on a


Alan's square
http://www.hiraeth.com/alan/misc/game/game.html
go
Professor Alan's puzzle square

Ok, so you've heard of Rubik's cube here's Alan's square. Click the arrows to move each column or row of squares round one step. Do a few at random, then try to get back to where you started. Try to get to the combinations at the sides of the screen. Click one of the demonstration combinations to reset the square.
Have Fun! Professor Alan

different squares to try ...
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
number square
Christmas
things to do ...
add a puzzle square to your own page
make your own puzzle square
coming soon
the mathematics of the square ... and how to solve it
other things

Figure 21.3 Interacting with hypertext – Professor Alan's puzzle square. Screen shot frame reprinted by permission from Microsoft Corporation

Macintosh HyperCard stacks can control applications using AppleEvents, or on a Windows platform hypermedia can include ActiveX components.

Whilst the 'text' in hypertext suggests passive content under the user's control, some hypermedia may contain more intelligent components or agents actively working to shape the experience for the user. For example, some educational hypertexts adapt their content depending on a model of the learner. Similarly e-commerce sites may suggest additional products to buy depending on your previous purchasing and browsing behavior.

21.2.3 Delivery technology

On the computer

Some hypertexts, in particular help systems, are downloaded or installed permanently on a computer. This has the advantage of instant access and such applications need not use a standard viewer but may include their own bespoke browsing software. However, with media-rich hypertexts containing substantial graphics, video and audio clips it may be impractical to store everything on hard disk. Also, for copyright protection, some systems will deliberately not allow themselves to be copied from their original distribution media.

Many hypermedia systems are supplied on CD-ROM. This has the advantage of reasonably large capacity (650–700 Mbytes), but access is slower than with installed systems. For highly dynamic material, such as educational media, a special player is installed; alternatively, material such as software documentation may use a standard format such as web pages.

DVD delivered material is interesting as it is not text enriched with video, but instead a movie that has been ‘made interactive’. As older computers tend to have only CD-ROM drives, it is not possible yet to supply, for example, tutorial material on DVD and know that it could be used on any machine.

On the web

Of course, the *world wide web* is the best-known multimedia hypertext system of all. The world wide web offers a rich environment for the presentation of information. Documents can be constructed that are very different from paper versions; basic text can be augmented through the use of hypertext links to other documents, while graphics can easily be incorporated as pictures, photographs, icons, page dividing bars, or backgrounds. Pages can also have hypertext links embedded into different regions, which take the user to a different page or graphic if they are clicked on; these are known as active image maps. These features allow web pages to become interactive, acting as the interface to the information as well as its holder. Dynamic material in the form of movies and sounds is also available to the designer; all these features push web page design well away from the conventional paper-based kind.

However, the fact that the web has many more technological features than a book doesn’t mean that web pages are necessarily better than their paper counterparts. Also, the fact that a web page is packed full of features doesn’t imply that it gets its message across effectively. Designing web pages is a developing art, and should be viewed in much the same way as designing any other interactive system. Good pages have been developed with the reader as the focus, and act as effective interaction tools or presentation tools to allow the user to obtain the information he is looking for most effectively.

The web is an example of the usability of the medium being paramount rather than the technological supremacy of certain aspects of it. For any single aspect of the web, there are custom multimedia systems that are far superior in terms of quality, usability, speed and so on. However, the common availability of web browsers on multiple platforms and the low-tech and usable nature of the system have ensured the web’s popularity and made it ubiquitous.

The web allows the user to browse documents and follow links transparently, with the underlying system taking care of the details of fetching the data from different parts of the world. Theoretically, as far as the user is concerned, any page can be reached as easily as any other; geographical location ceases to become important, whereas linking by content is crucial. The ability for anyone to publish information on the web is one factor in its success as a multimedia system, but the fact that anyone can create a page and, by linking it to others already in existence, immediately integrate their opinions seamlessly into the information space is another.

On the move

Mobile phones, PDAs (personal digital assistants), and notebook computers have all increased the demand to have hypermedia available on the move. Furthermore, across many countries governments have sold franchises for high-bandwidth mobile services. After spending billions on these franchises the telecommunications giants *really* want people to use new mobile services!

Notebook computers can use just the same mechanisms as desktop computers, using CD-ROM or DVD for standalone material, or connecting to the web through wireless access points or through modems linked to mobile phone networks. However, the fact that the computer is mobile means that location can be used as a key into context-aware hypermedia showing different content depending on location. The 'stick e note' system developed by the University of Kent uses a sticky note metaphor with notes stuck to particular locations [49]. Only when you visit the location does the note become visible. This is a bit like an image map on a web page, but rather than clicking a mouse over an image to link to information, here you need to physically move to a location! Another example is the GUIDE system, which uses various means to detect location (closest network access point or GPS) and then delivers appropriate tourist information [69].

PDA access poses different problems. They often have standard web browsers, but of course on a substantially smaller screen. This may mean designing special pages or being especially careful to design ones that resize well. Because PDAs are often not network connected there are also systems to allow access to information when disconnected. For example, AvantGo (www.avantgo.com) allows users to select 'channels' of interest and then, when the PDA is docked, downloads any pages that have changed, so that users always have the most up-to-date information possible.

By nature, mobile phones are (nearly) always connected to a network. However, memory and screen size are even more constrained. Some phones allow downloadable applets so that small dynamic applications can be used. More web-like content can be accessed via WAP (wireless application protocol), which, like HTTP (hypertext transfer protocol) for the web, gives access to remote servers. WAP content can be produced as static or dynamic content using a mark-up language called WML, which is a simplified version of HTML (hypertext markup language). This allows hyperlinks like the web and even simple images, but due to the small screen size most pages consist mainly of small amounts of information or simple lists of links. See the book WAP site at /e3/wap.wml

21.2.4 Application areas

There are many applications of hypermedia, too many to describe in detail here. However, it is worth noting the type of domains in which hypermedia systems have proved successful, looking briefly at some example systems.

Rapid prototyping

Although now lacking the wealth of features expected of a hypermedia system, HyperCard on Macintosh computers has been very influential as a basis for experimental hypertext systems. HyperCard uses the metaphor of a card index, around which the user can navigate. Each card can hold text, diagrams, photographs, bitmaps and so on, and hot-spots on the cards allow movement between cards. Cards may also contain forward and backward buttons and a home icon, to allow the user to move sequentially and start from scratch respectively. HyperCard can be used for a range of applications including information management and teaching.

However, HyperCard's simple scripting language and easy to produce graphical interfaces meant it was also used extensively as a *rapid prototyping tool* for generating interactive systems. In fact, HyperCard stacks for both single-user and networked applications are available from the book website.

For similar reasons, other hypermedia tools such as Macromedia Flash and Director are often used to produce dynamic interface mock-ups or even fully functioning systems. The web, too, is used like this, both to deliver applications and also as a way of mocking up an application interface as a series of storyboard web pages.

Help and documentation

Hypermedia systems are ideally suited to online manuals and other help system applications (see Chapter 10). They allow user-oriented access to the information, and support browsing. In addition the information can be organized hierarchically, with successive selections providing more detailed information. This supports the varying needs that users have, such as quick reference, usage information, full details and so on. Many commercial help systems use hypermedia-style help. Good examples are the Sun Guide system, HyperCard help and Microsoft Windows help (used by many Windows applications).

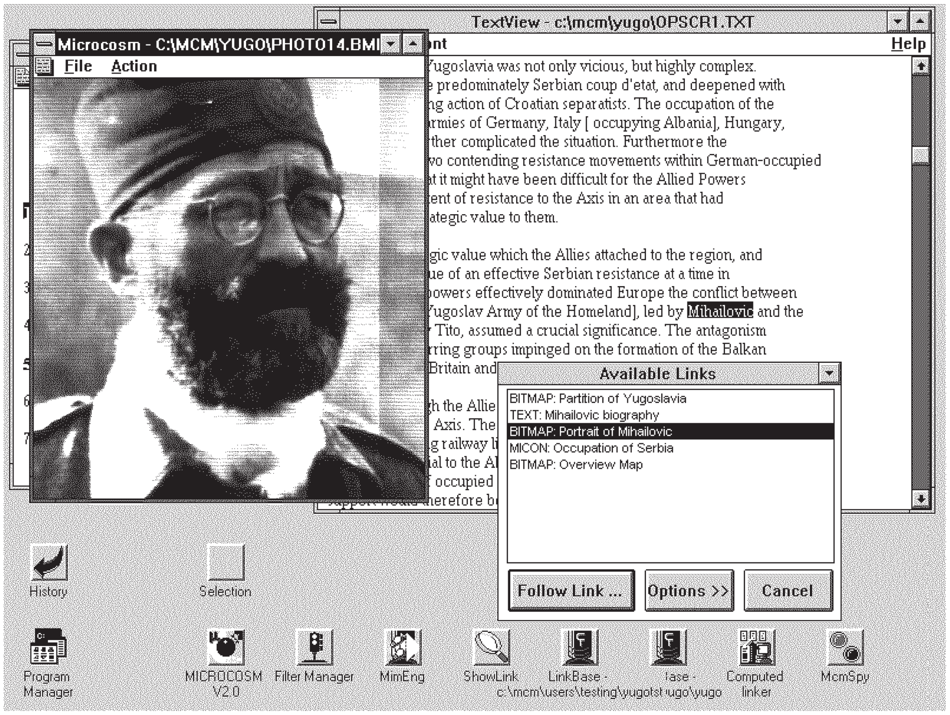
Educational systems are another common application. Hypermedia provides an environment for the learner to explore, in her own time and at her own pace. The inclusion of animation and graphics can allow the user to see things happen as well as read about them. So, for example, animation can be used to simulate an experiment. Educational applications are discussed in more detail in Section 21.6.

Education and e-learning

Hypertext and hypermedia are used extensively in educational settings, as they allow varied subjects to be related to each other in numerous ways so that the learner can investigate the links between different areas. In contrast to *computer-aided learning* (*CAL*) packages, hypermedia allows a student-controlled learning process.

An early example of a hypermedia learning environment that inspired many subsequent systems was Intermedia [385]. This is a hypermedia system built and used at Brown University to support teaching in subjects as varied as English literature and biology. The system includes text, diagrams, photos and so on. Both learners and teachers can add information and links, giving students access to each other's opinions as well as those of their tutors. A map provides an overall view of the information for direct access and navigation, with links providing browsing facilities in the normal way. Intermedia has been successfully used for university-level teaching, and can be seen as a forerunner of the educational resources now facilitated by the web.

Microcosm is an open hypermedia system, developed in the Electronics and Computer Science Department at the University of Southampton and shown in



Yugoslavia was not only vicious, but highly complex.
e predominately Serbian coup d'etat, and deepened with
ng action of Croatian separatists. The occupation of the
rmies of Germany, Italy [occupying Albania], Hungary,
rther complicated the situation. Furthermore the
vo contending resistance movements within German-occupied
at it might have been difficult for the Allied Powers
ent of resistance to the Axis in an area that had
ategic value to them.

gic value which the Allies attached to the region, and
ue of an effective Serbian resistance at a time in
powers effectively dominated Europe the conflict between
Yugoslav Army of the Homeland], led by Mihailovic and the
Tito, assumed a crucial significance. The antagonism
rring groups impinged on the formation of the Balkan
Britain and

h the Allie
Axis. The
g railway li
ial to the Al
f occupied
herefore b

Microcosm - C:\MCM\YUGO\PHOTO14.BMI
File
Action
TextView - c:\mcm\yugo\OPSCR1.TXT
nt
Help
Available Links
BITMAP: Partition of Yugoslavia
TEXT: Mihailovic biography
BITMAP: Portrait of Mihailovic
MICON: Occupation of Serbia
BITMAP: Overview Map
Follow Link ...
Options >>
Cancel
History
Selection
Program Manager
MICROCOSM V2.0
Filter Manager
MimEng
ShowLink
LinkBase - c:\mcm\users\testing\yugotst
ase - ugo\yugo
Computed linker
McmSpy

Figure 21.4 Microcosm: an open hypermedia environment shown running a teaching application entitled 'The civil war in Yugoslavia 1941–45' (developed by the Departments of Electronics and Computer Science and History at the University of Southampton and used by permission)

Figure 21.4; it allows users to browse through large amounts of multimedia information by following links. Microcosm does not contain links explicitly hardwired into its structure, but instead holds a database of link sources and destinations. Not only does this allow processes to examine the links divorced from their surrounding information, but it allows links to be made to read-only media such as CD-ROM, video discs and external web pages. Microcosm is able to integrate many different types of media, including other applications, and provides a set of viewing tools to look at text, bitmaps, video, audio and other component information media.

Outside organized educational settings, the increasing availability of multimedia PCs has made the home an attractive marketplace for CD-ROM-based educational software. One example of this is Dorling Kindersley's series of CD-ROMs based on their popular large-format book series.

As in other areas, the web has come to dominate most educational uses of technology. The term e-learning is sometimes used to refer to all forms of electronic learning, but is most often used for learning delivered via the web. This may be distance learning or it may be augmenting classroom teaching using the web.

The eClass project (previously called Classroom 2000) is an example of the latter [4]. It records pen strokes on an electronic whiteboard during a lesson (Figure 21.5) along with audio and video, and then allows students to review the material later

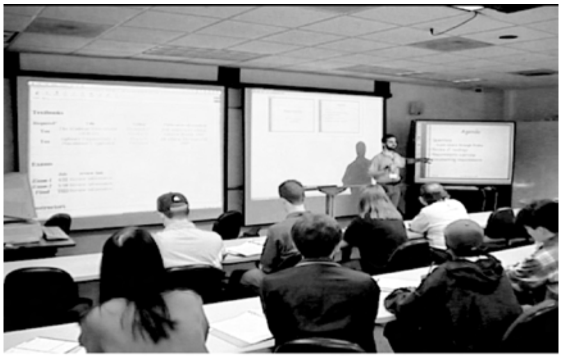


Figure 21.5 eClass – recording a lecture

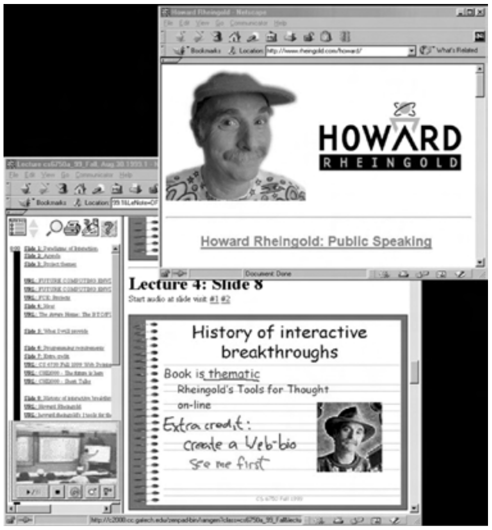

HOWARD
RHEINGOLD
Howard Rheingold: Public Speaking
Lecture 4: Slide 8
History of interactive breakthroughs
Book is thematic
Rheingold's Tools for Thought
on-line
Extra credit:
Create a Web-bio
See me first

Figure 21.6 eClass – indexed playback on student's web browser. Source: Netscape browser window © 2002 Netscape Communications Corporation, used by permission

using a web interface (Figure 21.6). Each slide presentation and pen stroke is time-stamped so that students can click on a particular pen mark on the web interface and the audio or video is replayed from that point in the lecture.

Collaboration and community

Although strictly not hypertext, the web has become a central platform for collaborative applications and community. These use the hypertext structure of the web to structure and access shared resources and message areas. For example Yahoo! Groups (groups.yahoo.com) allows mailing lists, shared images (such as family photo albums), web archives of the mailing list and chat, all accessed through a web interface.

Establishing a sense of community can be very important on websites as it is one way to ensure loyalty and get visitors to return. This may involve explicit community features such as chat areas, or may simply be a matter of using a design, language and image that suggests a site which is open and listening to 'readers' [214].

E-commerce

For some companies the web is simply another sales opportunity. Many readers will have used online stores such as Amazon or bought from an auction site such as eBay. Hypertext's use of hierarchies, links, images and so on, makes it ideal for displaying certain kinds of product. Actual buying and selling requires not only security at the level of the networks, websites, etc., but also trust. When you walk into a shop you can see the person you are dealing with, and the fact that it is physically there today gives you confidence that it will be there tomorrow if anything goes wrong. How to build and ensure this trust is an active area in HCI research (see, for example [307]).

21.3 FINDING THINGS

21.3.1 Lost in hyperspace

Although the non-linear structure of hypertext is very powerful, it can also be confusing. It is easy to lose track of where you are, a problem that has been called 'lost in hyperspace'. There are two elements to this feeling of 'lostness'.

The first is cognitive and related to content. In a linear text, when a topic is being described, the writer knows what the reader has already seen. In a hypertext, the reader can browse the text in any order. Each page or node has to be written virtually independently, but, of course, in reality it cannot be written entirely without any assumption of prior knowledge. As the reader encounters fragmentary information, it cannot be properly integrated, leading to confusion about the topic.

The second is related to navigation and structure. Although the hypertext may have a hierarchical or other structure, the user may navigate by hyperlinks that move across this main structure. It is easy to lose track of where you are and where you have been.

The solution to the former issue is to design the information better. The solution to the latter is to give users better ways of understanding where they are and of navigating in the hypertext. To say 'the solution' is disingenuous – there is no simple 'solution'. If we want to provide information that allows complex, unplanned, non-linear access, there will probably always be problems. However, good design can help!

21.3.2 Designing structure

We discussed the importance of good structural design in Chapter 5. Some of the task analysis techniques presented in Chapter 15 may be useful in giving a task-oriented or knowledge-orientated breakdown of a hypertext. In some areas there may be pre-existing understood structures to mirror; for example, the faculty and departmental structure of a university, or the main disciplines (circulatory, neurological, etc.) within medicine.

In a paper format one is stuck with a single structure, which can lead to tensions: for example, the fact that in this book structural design is discussed in several places. As another example, imagine a car mechanic using a manual. She might want to use the classical breakdown into transmission, fuel system, etc., while fault finding, but if she were dismantling the engine it might be more useful to look at the car components in terms of physical location.

If multiple structures are used, you have to consider what to do about the common material. For example, if we examine a car hypermedia text under 'engine compartment' and get to the fuel pump, this would also appear in the functional view under 'fuel system'. Such common elements may be replicated. This has the advantage that the material can be presented in ways that make sense given their context, but it can also lead to inconsistencies.

Alternatively we may make links across the hierarchy at some level; for example, the engine compartment may have a diagram of the engine with a labeled arrow saying 'fuel pump (fuel system)', which takes you to the description of the pump in the fuel system part of the hypertext. Notice, too, the importance of making links that go to different parts of the hypertext very clear, following the 'knowing where you are going' principle from Chapter 5 (Section 5.6.1).

An in-between solution would be to have a small dedicated description of the fuel pump in the context of the engine compartment: perhaps describing how it is physically connected to the engine block and how the piping is routed. This could then be linked to a more extensive explanation of its function in the fuel system section, using a 'see also' link.

In all cases it is important that the structure and the naming of parts is meaningful for the user. In a more detailed and theoretical approach to the 'knowing where you are going' principle, Pirolli and others have developed information foraging theory [289]. This uses an analogy with foraging animals searching for patches of food and trying to make decisions about when to move to a different area or stay with the food available, and, if they move, where to go. This is likened to the way an information seeker browses, making decisions about whether to stick with the information available or spend time looking for more, and, if more is needed, deciding where to seek it. For animals, the scent of food is very important in deciding where to go, and information foraging theory suggests that it is crucial that our structures, access methods and detailed page designs give users some sort of scent – hints about what they may find – to allow them to make good browsing decisions.

21.3.3 Making navigation easier

No matter how well designed the site structure is, there will still be problems: because the user does not understand the structure; or because the user has individual needs that the designer has not foreseen; or because even a good structure is not perfect. However, there are various things that can make it easier for users.

One solution is to provide a map of the hypertext document, identifying the current position of the reader within it. Links to home or end points can then be identified and the user is less likely to get lost. This may be a separate part of the hypertext; for example, some websites have a site map link leading to a special page, and many help systems have a table of contents view. Alternatively, the site map can be woven into the layout of the document; for example, some sites have an outliner-style sidebar listing the main sections and drilling down to the current location. This acts both as an indication of where you are in the site (like the breadcrumbs discussed in Chapter 5) and as a constant reminder of the overall site structure.

One way to deal with the differing uses that are envisaged for a hypertext is to provide the user with ‘recommended routes’ through hypertext documents. If these are too strongly defined they defeat the objectives of the system. However, successful applications have been developed that have a guided tour or bus tour metaphor [13]. Users can follow the tour, which takes them through one or more selected routes, but can deviate from it at interesting points whenever they wish. Users can then return to the ‘bus stop’ to continue the tour when they have satisfied their interest.

Another type of hypertext takes the form of ‘levels of access’ to a document. Different levels of access privilege ‘see’ different amounts of information. A document structured in this way may provide one level of access that gives only a brief overview of the topic. The next level of access presents a fuller description of the system, while the next level may also include information regarding the precise meaning of technical terms used in the system. The final level of access may add historical information and suchlike. The user can choose at which level he wants to read the document, cutting out irrelevant information while obtaining all the necessary details. Such a document tends to be linear in nature, which makes navigating and printing it easier, but removes the user’s choice in structuring his progress through it.

Once information has been retrieved, a paper version is often needed. Printing a document requires the pages to be in a particular order, but hypertext does not support the concept of one single order. This is against the ethos of hypertext, which intends the user to structure the information in the way that suits him best. It can therefore be difficult to get a hard copy of the information that is required.

Although there is no simple way to linearize a hypertext, one can at least make it possible to print individual linear parts, whether single pages or groups of linearly linked pages. In general, you should not rely on the print facility of a browser as this is printing a page designed for on-screen viewing. You may notice websites offering printer friendly pages. These may be in a different format such as PDF, or may simply be web pages without sidebar navigation aids, etc.

For both web and non-web hypertext, remember that once the page is printed someone who hasn’t seen the full hypertext may read it. If they like what they read, can they find the source? If a web page is likely to be printed, do include the full URL on the page, perhaps in the footer. On pages from a CD-ROM or similar hypertext, include perhaps a copyright notice with details of the original.

21.3.4 History, bookmarks and external links

Hypertext viewers and web browsers usually have some sort of history mechanism to allow you to see where you have been, and a more stack-based system using the ‘back’ button that allows you to backtrack through previously visited pages. The back button may be used where a user has followed a hyperlink and then decided it was to the wrong place, or alternatively, when browsing back and forth from a central page that contains lots of links. The latter is called *hub and spoke* browsing. In fact, in studies of web browsing the back button accounted for 30% of all navigation actions [63]. Other studies have shown frequent revisiting of pages during a single browsing session [341].

Although the back button is used extensively, it is used relatively little to go back more then one step. For error correction this makes sense, but for general revisiting one might think that moving back several steps would be common. Possibly, one reason for this is confusion about the meaning of the back button; indeed a formal comparison of back and history mechanisms in four different hypertext and web browsers found that the operation of back and history were subtly different in each [104].

For longer-term revisiting, browsers typically support some form of bookmarking of favorite pages. Both this and, on the web, external links from other people’s sites mean that users may enter your hypertext at locations other than the top level or home page. On the web this is called *deep linking*. Many websites rely on the user remembering where they have come from to make sense of a page. If a page does not adequately show where it fits, then a user coming to it from outside may have no idea what site it is from, or why they are reading the material. Furthermore, if the original site depended on the user pressing ‘back’ to return to higher levels of the site hierarchy it may be impossible for a visitor to find the rest of your site at all!

All pages should therefore make clear where they belong and have links into the full site structure. For example, the breadcrumbs along the top of a web page function both to make the context clear and, if they are ‘live’, as navigation back into the site hierarchy (see Chapter 5, Section 5.6.1).

Framed websites are particularly difficult. The material you want to bookmark or link to may be one of the frame content pages, but the URL you can see or bookmark is that of the overall frameset. This may either discourage linking to the site or, if circumvented, it may mean linking directly to the content of one of the frames, which is then very likely to lack sufficient context, being designed to be seen within the frameset. Search engines, too, may generate links to individual frames in a frameset. Many web style guides heavily discourage the use of frames for this reason. If the site is designed using a development tool that supports page templates, or is being dynamically generated, there is rarely any need for frames as most of the effects can be obtained using other page formatting.

Very occasionally you may want to discourage deep linking; for example, if the framed page is more of an interactive application or you know the inner structure is unlikely to stay constant. In such cases you can include a small piece of script in the inner framed pages that makes them redirect to the outer frame if they are ever opened 'bare' in a window. This means that if a site or a search engine does link into the inner frames, following the link takes the user to the full, framed site.

For more on usability issues of frames see /e3/online/frames/

21.3.5 Indices, directories and search

As well as a hierarchical table of contents structure, many help systems, hypertexts, and for that matter paper books, have some kind of index. Note that an index is not a complete list of all words in a document. If this were the case then the index for this book would be as big as the rest of the book! The words in an index are chosen because they are significant key phrases or words with a domain meaning, and not every occurrence of a word is indexed, only those deemed in some way important. The main difference between an electronic index and a paper one is that with the paper index you have to physically look up the page after finding the word in the index, whereas in an electronic index the links are 'live' so you can simply click to the content.

On the web an index would be very big (!); however, directory services such as Yahoo! (www.yahoo.com) or the Open Directory Project (ODP) (www.dmoz.org) can be seen as a form of index. The main difference is that while an index is simply an alphabetical list of keywords, web directories give a hierarchical categorization to sites. The categorization is done either by self-submission, or, in the case of quality directories such as ODP or Yahoo!, by experts in the relevant field.

For exhaustive searching by keywords, some kind of automated search is required. In the case of a standalone hypertext, the viewer application may do this either by using a pre-computed electronic index of all word occurrences used by the hypertext, or by scanning it on demand. The latter will take longer for each search, but may be more effective if the hypertext is not too big or the material is rapidly changing. Where the hypertext is generated from a database, the search may be performed on the underlying data rather than the generated pages.

In the case of the web the content is dynamically changing, but it would be impossible to scan the whole web every time you wanted to find anything! Search engines such as Google or AltaVista use web crawling. Starting from an initial collection of pages they look for all links from these pages. These links are followed and the new pages reached are scanned, and so on. As pages are visited, an index is built of which words occur in which pages. The search engines typically do not keep a copy of every page visited, but may just keep the title and the first hundred or so words on each page. Even the index is vast and so the most common words are usually not indexed; these are called *stop words*. When you do a search, the search engine uses the index and the summary information to construct the results page with links to the actual pages. Because it is using the page summaries and not looking at the pages themselves, it is possible that a page may have been removed or changed since the index was constructed.

The web is enormous and so the number of pages containing a given word is enormous. Web search engines allow you to search for several words at once or for exact phrases, or, with Boolean searches, to specify using logical and/or options what is required: for example, 'engine AND NOT car'.

Even when looking for multiple words or Boolean queries, the number of results may be in the tens or hundreds of thousands. So search engines need some way to rank pages. Some use simple, content-based measures such as the number of times the requested words occur, whether they occur in the title or body, whether they occur near the beginning or end of the page. Some search engines keep track of how many times users click through for specific pages and so can build up a model of popularity. In addition, some search engines sell the right to be top, based on keywords, or have a special advertisers or sponsors links section.

Google gained its reputation for quality based on the way it ranks pages, and seems almost magically to get just what you want in the first page or so of results, and often as the very top ranked result. It does this by using the structure of links. It uses heuristics such as: if a page is linked from lots of places it is likely to be good, especially if the pages linking to it are already known to be good; and also the opposite: if a page links to lots of good pages then it is a good page.

Some specialist searches, for example for video, books, etc., and even some more general-purpose search engines, allow you to rate pages you have visited. Ratings are then used to rank the more popular pages for future visitors. These are called recommender systems [304]. As well as explicit recommendations, e-commerce sites often track your browsing on the site and use this to build a profile of which users are similar to you. The books or goods they purchased may then be suggested to you.

When designing web pages, it is possible to make them more 'search engine friendly' by adding 'META' tags in the head section of the web page, in particular keywords and description, as well as a relevant 'TITLE' tag. In the early days, people tried to fool search engines by including invisible lines with lots of popular keywords at the top of their document. Now most search engines are able to detect such subterfuge and discard such pages – so be honest! However, it is worth making sure that something near the top of your page says what it is about!

Search engines have trouble scanning sites with many generated pages, especially if they are accessed through a search box only, for example a dictionary or thesaurus. There are many large data sets available on the web, often public domain or freely accessible, which contain high-quality information – often better quality than any old web page – but are not easy to find unless you know the site. This has been called the hidden web and some estimates say that it is an order of magnitude larger than the visible web. Currently, there are no accepted ways to link such material into broader web searches although some products, such as onCue described in Chapter 18 (Section 18.4.1) attempt to link into these publicly accessible data sources.

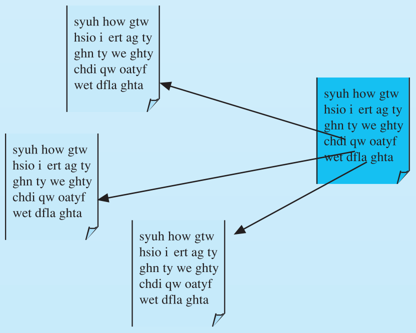
Searching for research literature

When you are asked to do a project or a literature review you will probably reach straight for Google! You will get many pages that are not relevant and many that appear to be, but how do you know which are worth reading? Unless, of course, you just assume that Google knows best!

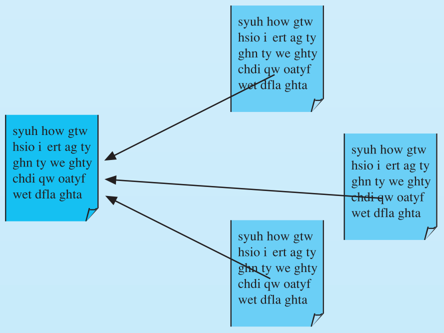
Part of the process by which papers are accepted into a conference or journal is peer review – the papers are read and judged by other workers in the field. On the web, it is hard to judge the quality of information you find.

There are some heuristics you can use. Is the online page an electronic version of an article published in a refereed paper publication? Where is the page: in a major corporation or university, or a free home page? This doesn't mean that you can't find good pages by unknown people, but existing position is a heuristic. If the author has a publications page, are other articles published in good venues? Do other pages that you have judged to be of good quality reference an online article? And, of course, not to be forgotten . . . does it make sense? No matter who wrote it, if it is not well argued then don't believe it!

If you are looking for articles in computing you can also use *citeseer* (citeseer.jn.nec.com). This is a search engine for online copies of papers, hosted by NEC Corporation. Unlike standard web search engines, this only searches for articles, and in particular is able to scan PDF, PostScript and other electronic formats. Not only does citeseer search by title, author, etc., it is also a citation index. For each paper, it scans it to find the bibliography at the end. When you look at the entry for a paper, it shows you what other papers it references (looking back in time) and also which further papers have cited it (looking forward), as the diagram shows.


syuh how gtw hsio i ert ag ty ghn ty we ghty chdi qw oatyf wet dfla ghta
syuh how gtw hsio i ert ag ty ghn ty we ghty chdi qw oatyf wet dfla ghta
syuh how gtw hsio i ert ag ty ghn ty we ghty chdi qw oatyf wet dfla ghta
syuh how gtw hsio i ert ag ty ghn ty we ghty chdi qw oatyf wet dfla ghta

Bibliography backwards in time


syuh how gtw hsio i ert ag ty ghn ty we ghty chdi qw oatyf wet dfla ghta
syuh how gtw hsio i ert ag ty ghn ty we ghty chdi qw oatyf wet dfla ghta
syuh how gtw hsio i ert ag ty ghn ty we ghty chdi qw oatyf wet dfla ghta
syuh how gtw hsio i ert ag ty ghn ty we ghty chdi qw oatyf wet dfla ghta

Citations forward in time

21.4 WEB TECHNOLOGY AND ISSUES

The web has featured strongly already in this chapter and for the remainder of the chapter we will focus exclusively on it.

21.4.1 Basics

The web consists of a set of protocols built on top of the *internet* that, in theory, allow multimedia documents to be created and read from any connected computer in the world. The web supports hypertext, graphics, sound and movies, and, to structure and describe the information, uses a language called *HTML* (hypertext markup language) or in some cases, XML (extensible markup language). HTML is a markup language that allows hypertext links, images, sounds and movies to be embedded into text, and it provides some facilities for describing how these components are laid out. HTML documents are interpreted by a viewer, known as a *browser*; there are many browsers, and each can interpret HTML in subtly different ways, or support different levels of functionality, which means that a web page viewed through one browser can look very different from the same page viewed through another. The web requires no particular multimedia capabilities from the machines that run the browsers; for example, if sound is unavailable on a particular machine, then obviously no sound is heard but the browser still displays the text happily.

The web owes its success to many factors, including the robustness and (relative) ease of use offered by popular browsers from the very first graphical browser *Mosaic*, and continued in commercial browsers such as *Netscape Navigator*, *Microsoft Internet Explorer* and *Opera*. These offer a graphical interface to the document, controlled by the mouse. Hypertext links are shown by highlighting the text that acts as the link in an alternative color, and are activated by clicking on the link. A further color is used to indicate a link that has already been visited. Hypertext links can also be embedded into regions within an image.

Although the browser contains most of the functionality required to view a web document, supporting text and graphics in an integrated package, special file formats and media, including some movie formats, may require additional plug-ins or helper applications.

As well as publishing personal, corporate and governmental information, the web is used as a source of entertainment, an advertising medium, a communication environment, and more. The vast and ever-increasing quantity of information available on the web certainly exacerbates the user's 'lost in hyperspace' problems (Section 21.3.1). But increased familiarity with hypertext, and better web page design, are aiding the situation at least as much as technological efforts to create maps and indexes of the data. Another problem is that of information overload: multimedia images, gigabytes of graphics and mountains of text swamp the reader in a glitzy but unmoderated world, in which the fact that almost anyone can make anything available leads to the gemstones often being lost amongst the slag.

As well as static web content such as text and images, many pages are *dynamic*: for example, they may be generated from data held in databases, respond to individual information entered into forms, or include dynamic elements such as Java applets. We will discuss this dynamic content further in Section 21.6.

21.4.2 Web servers and web clients

Whereas a conventional PC program runs and is displayed on one computer, the web is *distributed*. Different parts of it run on different computers, often in different countries of the world. They are linked, of course, by the internet, an enormous global computer network (see also Chapter 2, Section 2.9.3).

The pages are stored on web servers that may be on a company's own premises or in special data centers. Because they are networked, the webmaster for a site can upload pages to the server from wherever she is. For example, the web pages for www.hcibook.com are stored in a data center several thousand miles from where any of the authors live!

Your machine, the PC running the web browser, is called a *client* because it wants the pages from the servers. When you click on a link your web browser works out the full URL of the page it needs: say 'http://www.hcibook.com/e3/authors.html'. It splits this into parts. The first part is the protocol 'http' which says how it talks to the server (other alternatives include 'ftp'). The second part 'www.hcibook.com' is the host name, that is the name of the web server containing the requested page. The last part '/e3/authors.html' gives the particular file on the site. The browser then establishes a connection to the required web server (in this case 'www.hcibook.com'), and sends a message, formatted using the HTTP protocol, to the web server, which then finds the requested html file (or image, or other file type) and returns it to the browser, which then displays it to you.

If the page contains images the same process is repeated for each image, and if the page is a framed one for each frame within the page.

21.4.3 Network issues

The fact that the web is networked raises a series of issues that can impact on usability.

Network capacity is called *bandwidth*. This is a measure of the amount of information that can pass down the channel in a given time. For example, a typical modem speed is 56 kbs – that is 56 kilobits per second. This equates to about 6000 characters per second. This sounds fine until you realize that images may take many tens or hundreds of characters (bytes) to encode . . . this is why many have renamed the web the 'world wide wait'!

However, bandwidth is not the only important measure. There is also the time it takes for a message to get across the network from your machine to the web server and back. This delay is called *latency*. Latency is caused by several factors – the finite speed of electrical or optical signals (no faster than the speed of light), and delays

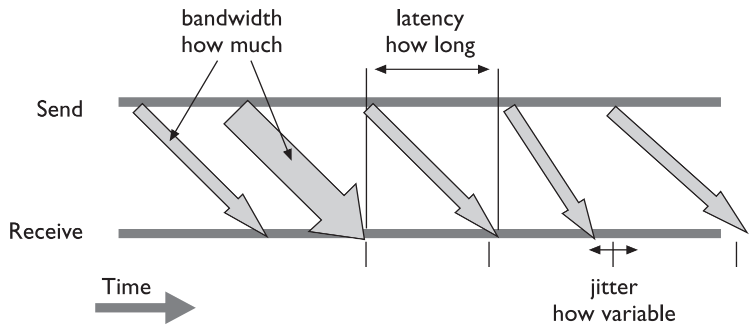


Figure 21.7 Bandwidth, latency and jitter

at routers along the way that take messages from one computer network and pass them on. This latency may not always be the same, varying with the exact route through the network traveled by a message, the current load on the different routers, etc. Variability in the latency is called *jitter* (see Figure 21.7).

As well as the underlying latency and jitter of the network, each layer of network software adds its own. The underlying internet protocols are *lossy*, that is messages can be lost. When this happens, higher level software (the TCP – transmission control protocol – layer) notices and resends messages to give reliable communication. These losses mean that the average delays increase, but also that the jitter increases – a lost and resent message takes more time than one that gets there first time.

There may also be an appreciable amount of time setting up a new connection, which may outweigh the time taken to actually send data. This is particularly a problem with sites containing many small images.

Web usability guides will give plenty of advice on ways to minimize web page size, but one recommendation, which sadly we promulgated too in earlier editions of this book, should be taken with a pinch of salt. Virtually every guide suggests keeping pages short – enough to fit within a single screen of a typical web browser. Now there are arguments for this – some users may not realize there is more content off the bottom of the screen, the material lower down may be ignored, search engines are better able to index multiple small pages. However, the reason that is often given, namely download time, is not a good one. Of course, an enormous page takes a long time to load, but on the whole interaction is faster if a larger page is downloaded and then scrolled through than if the user has to click links on many small pages and wait for each.

This latter point is an example of a broader issue. In general, if you want rapid *feedback*, try to make the interaction happen at the web browser end. Feedback that involves a cycle to the web server and back, except for intranets on fast networks, will take several seconds, which is too slow for fluid interaction. Unfortunately, for collaborative applications there is no way to prevent at least some network delays for *feedthrough* (see Chapter 19, Section 19.6.3), but even then it is important to prevent too many cycles of network messages.

Delivering WAP content – balancing usability and feedback

Because of the tiny screens on phones it is difficult to scroll through a long WAP page. However, if every link involved going back to the WAP server the feedback would be very slow. For this reason WML divides WAP content into *stacks* of *notes*. For the user browsing the content, the note is the parallel of the normal web page. A link may be to a note in the same stack or one in a different stack; the user is largely unaware of which it is. However, when you request a note in a stack the whole stack is downloaded to the WAP browser on your phone. This means that links to notes in the same stack have much faster feedback. By carefully arranging content within stacks and notes, the overall user experience can be improved.

21.5 STATIC WEB CONTENT

21.5.1 The message and the medium

One thing is often forgotten when web pages are created. It is of vital importance, and hence will be discussed first.

It is content.

Many people assume that because they can make information available on the web, they should. Unfortunately, because it is very easy to publish information, much less care is taken with the actual content. Material may be nonsense, it may be incorrect, it may not read well, or be incomplete, or inane.

Excellent page design can make useless material look attractive, but it still remains useless material. On the other hand, poor design can mean that excellent material is never seen by potential readers, as they have become bored, or intolerant of the medium, or confused, or for a host of other reasons have aborted their attempts to download and view the information. Pages do have to look immediately interesting and attractive if people are to spend time, effort and, because of the communication costs, money, in viewing them; the user-centered nature of the medium makes this imperative. This is in marked contrast to television or cinema or other dynamic media, which are not under any direct user control, where information is presented to a passive audience. With web documents, people have actually to want to see the information, and make an effort to retrieve it, which clearly must have an influence on design.

Whatever is being presented, underlying all the comments made on good and bad design, the fundamental message is that, for the user group or groups who are targeted, the content should be worth reading.

When it is likely that a user will require a paper copy of the information made available over the web, ideally they should be able to download it in one go as a single complete file, with the same information content but possibly a different layout. Paper does not have the same inbuilt hypertextual and active capabilities as the web page, and will be accessed in a predominantly linear fashion.

21.5.2 Text

Because web pages are displayed on many different machines, there are only a small set of fonts that can be guaranteed to be available: a standard font and a typewriter font (e.g. `courier`) with bold and italic versions in different sizes. However, it is possible to specify preferred fonts and many of these such as Arial, Verdana or Comic Sans are available on most web platforms. The difficult thing is to balance fine tuning the appearance of the text on one platform with making it readable on all.

The various structured styles such as headings allow the web designer to create material that will lay out passably on all platforms. But these offer a fairly coarse level of control. The size and boldness of the heading should be chosen carefully; for example, huge dark fonts on a page can look loud and brash.

There is an increasing desire to have fine control. Cascading style sheets (CSS) allow you to specify fonts, line spacing, size, etc., in a similar way to styles in a word processor or DTP package. However, care must be taken. For example, many pages specify fixed point sizes that may not display well on different platforms and can cause problems for people with visual impairments.

The use of color is of great importance for web pages, but it is often abused. First, it should be remembered that a significant proportion of the potential viewers of the page will have problems with color, either because they are using older machines with a limited color palette, or because they have some form of color blindness. Color, when used, should not be the only cue available. Users also bring a deep-rooted emotional interpretation to colors; as we have seen, in some cultures, red is associated with danger and anger, whilst green is regarded as go, or safe. Blue can be a cool color, orange a warm one, and so on.

Links usually change color once they have been accessed, providing cues to the user about what material they have already explored. This means that two distinct but still suitable colors need to be associated with each link, so that the system is acceptable whether or not the links have been activated. Note, too, that consistent use of color can help the user understand the role of particular elements more intuitively, whereas color used for no clear purpose is often distracting.

One common mistake is to put colored text onto a similar colored background so that it becomes nearly invisible. One of the authors had a student who designed a beautifully laid out page of text, and decided to add a background to the page just before demonstrating it to the rest of the group. It was only at the demonstration that he realized that the cool black background he had added made the black text impossible to see!

There are only a limited number of text-placing options: text can be left or right justified, or centered. There are a few predefined formatting styles such as ordered and unordered lists that have additional structure, in the form of indentation from the left margin, with numbering in the case of ordered lists. Vertical positioning is even more limited, but tables and (cautious use of) frames allow a greater degree of horizontal and vertical placement. More precise positioning still can be obtained using 'dynamic HTML' (DHTML), which allows parts of an HTML document (called layers or 'div' sections) to be positioned as if they were separate mini-pages within the browser window. The word 'dynamic' is used because these can then be controlled using JavaScript to produce various animated effects (those pages where little stars follow your mouse!).

Remember that monitors are different sizes and that some people use full-screen windows and others smaller ones. To prevent very long lines, many designers lay out pages within tables that put maximum widths (in pixels) for columns based on typical minimum expected monitor sizes (perhaps 800×600 or even 640×480). If fixed layouts or large graphics are used then they may either display strangely on smaller windows or force the user to scroll horizontally, which many users find confusing. And don't forget people viewing the page on PDAs!

The lack of explicit textual positioning makes it very difficult to produce complex mathematical equations, and the font set available is not rich enough to provide a suitable approximation. Developments in the specification are addressing this, though the intrinsic complexity of typesetting mathematics suggests that it may be a while before a simple, usable solution is found that is acceptable to readers, page designers, and implementors of web browsers alike.

21.5.3 Graphics

Obtaining graphics

There are a number of sites on the web that contain archives of graphical images, icons, backgrounds and so on. There are also paint and image manipulation packages available on almost all computer systems, and scanners and digital cameras, where available, enable the input of photographs and diagrams.

Using graphics

While graphics and icons tend to play a significant role in web page design, their use should be carefully thought out. Graphical images take longer to load than text, and this may become a problem. Text uses 8 bits to represent a character: some rough calculations show that approximately 2000 characters represent about a screenful of information, and so 16,000 bits (2 K) are required. For graphics, one pixel may use 8 bits to represent its color: a page-sized image will be at least 600 by 400 pixels, which will take 1,920,000 bits (240 K), or 120 times as long to load. Put another way, while a picture may tell a thousand words, it takes approximately 50 times as long to appear! Users become bored with operations that take a long time to complete, and are unlikely to wait for ages while a page appears.

Complex backgrounds are the worst offenders in this area; they offer little in the way of added value to the information presented on the page, and cause great frustration for the poor reader. They tend to be designed and tested only on local machines, with high-bandwidth connections between them, which means that the time factor is negligible for the designer/user. However, this disregards the fact that many people accessing the page will be using congested, slow networks, with a transfer rate sometimes down to a few kilobits per second, rather than fast megabit links. Fussy backgrounds also have the unfortunate ability to obscure text, making it very difficult or impossible to read.

Different browsers support different types of functionality, with more recent versions having features that try to alleviate the usability problems introduced by the delay involved in downloading graphics. Most browsers support caching, in which graphics are downloaded once and temporarily stored on the user's local machine. If the same image is reused, it is fetched from the local store far more rapidly than if it were retrieved from the remote site. This clearly has implications for page design: if graphics are to be used, then their reuse wherever possible speeds up the whole process of drawing the page. Complex graphics can sometimes be broken down into a set of items, many of which can be reused and assembled in different ways to add visual impact to the page without causing large delays. Most browsers also offer the option of turning off automatic image loading, so that only the text is downloaded. If a page then appears to be of interest, the graphics can be explicitly requested. It is sometimes possible to set out a page so that it still looks attractive even without the graphics, which is necessary if the user has turned off image loading. There are other browsers that are purely text based and do not support graphics of any sort, and for these HTML offers an additional image attribute that allows a textual description of the image to be used as an alternative. The need to support these different user preferences and browser capabilities provides a great challenge in designing pages that are acceptable to all.

Some browsers have additional features related to image handling as a technological response to the problem of page usability. If the designer specifies the size of the image in advance, the browser can lay out the text on the page first, leaving spaces for the images. This allows the user to continue to read the page contents whilst the images are being downloaded into their respective slots. This capability improves the usability of the page, and so should be supported by the page designer whenever possible, by incorporating the necessary information into the image reference.

Both GIF (graphics interchange format) and JPEG (Joint Photographic Experts Group), the most widely used web graphic image formats, can be saved in forms that allow them to be progressively transmitted. This means that images appear as a whole, but very blurred, version that becomes gradually sharper, rather than appearing in perfect resolution a line at a time. An overall impression of the page and the graphic information appearing is thus given to the user, who is then better informed about whether or not to continue the download.

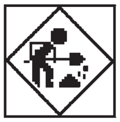
The JPEG format is optimized for photographic images and makes use of their properties to offer a higher compression ratio and hence faster loading. However, its compression is *lossy*, that is the image reproduced is slightly different from the original, losing certain kinds of visually indistinguishable colors and, more importantly, losing high-frequency change. The latter is because photographs tend to have slowly varying changes with few sharp edges. If sharp-edged images such as diagrams or text labels are stored as JPEGs, small *artifacts* are produced such as ripples appearing around letters and lines. In contrast GIF uses a *lossless* compression so that the image appears exactly as it started. Although GIFs can be used for photographic images, the compression is very poor.

The GIF format also allows *animated GIFs*. These are a sort of mini-slide show or movie where several images are stored in the same file and play one after another. These can be used to produce simple and effective animations, but when overused can lead to very 'noisy' pages.

Active image maps are pictures with defined 'hot' areas, which, when clicked, execute a particular script, usually calling another web page or graphic. The careful use of such maps can transform an interface, but there is an overhead to pay in loading the map and calling and running the script, and this should be considered carefully by a page designer. Another characteristic of image maps is that there is rarely any indication of which areas are active and which are not, and it can be difficult for users to ensure that they have visited all the active regions. For accessing spatially organized information, image maps are very suitable (for example, in a tourist information system, the area about which information is available can be represented well), but for information that does not have any clear spatial analog, their use is more questionable.

Icons

Icons often appear on web pages, and while there are many available to choose from, they should be used with care. On web pages, icons are typically used in one of two ways. They are either visual cues, associating some small picture with different parts of the text (for example, some pages have icon-sized characters that appear next to instructions). Alternatively, they are used in much the same way as in a standard WIMP interface to represent aspects of the functionality of the underlying pages. In this latter case, they must represent their associated functionality in either a concrete or an abstract form. This means that the design of the individual icon has to be carefully thought out, as a lot of information may have to be represented in a small area of screen estate. However, icons are rarely seen on their own, and when placed next to their neighbors, the whole effect has to be pleasing rather than disruptive and garish. Therefore, the group of icons has to be designed together, with a coherent and recognizable style. The picture is broader than this, however: other applications also use icons, which has its advantages and disadvantages. One advantage is that certain icons are already associated with specific functionality (for example, a picture of a floppy disk to represent 'save'). Disadvantages are that it restricts the individuality and style icon sets can show, and may mean that icons designed for one purpose are misunderstood by users because they have seen something similar in another context. It is therefore vital that time is spent in examining the way icons are used in other systems, before importing them into web pages or designing new ones.



Icons sometimes appear for no apparent reason at all, when the page creator has decided that as graphics are supported, a few should be used. One interesting example is the icon that is a copy of the roadworks sign, used in conjunction with text saying something like 'This page still under construction!'.

This is an interesting social effect brought on by the ease of web publishing – incomplete (sometimes even non-existent!) information can be made immediately available. It's like buying a map of the world nowadays and finding 'here be dragons' around the edges because the geographer could not be bothered to draw all the countries in. Most pages can be designed properly before they are made available, structured and presented in a complete and coherent way, allowing for extensions and updates from the beginning.

There are times when the disclaimer 'under construction' has its uses: when critically important information becomes available, publishing it may well be more important than presenting it (much like older maps with their dragons – if maps had only been printed once the whole world had been explored, civilization would be very different today). There is, too, a sense in which web pages can be continually 'under construction', changing, evolving and growing, because of their dynamic nature and the ease with which they can be updated, but this does not obviate the designer's responsibility to create pages that have both form and content.

Graphics and color

Using many different colors within graphics may well result in the browsers for older machines running out of entries in the colormap, with unpredictable consequences. This is often problematical as the browser may be running in tandem with other color applications, and only has a restricted range of colors to begin with.

For many consumer markets, for example in the UK and the US, this is unlikely to be a problem as home machines are often relatively recent. However, many businesses continue to use older PCs so long as they 'do the job' and PDAs may not have a full color palette. Furthermore, in economically deprived areas, where there is computer access it may well be through older or second-hand machines.

If universal access is required it is therefore still wise, where possible, to restrict images to a limited number of colors, taken from the standard 216 color web palette, and to reduce complex color images to simpler approximations. Reducing the number of colors used also allows the depth of the images to be reduced; a change from a default of 8 bits to, say, 4 bits will produce a twofold speedup in image loading. The earlier comments on the use of color obviously apply as much to graphics as they do to text.

One further point should be made about graphics: computer screens are typically limited to a resolution of around 72 dpi (dots per inch), and so either high-resolution images will have to be displayed much larger than actual size, or the increased resolution will be forfeited.

21.5.4 Movies and sound

Movies and sound are both available to users of the web, and hence to page designers. One problem associated with them is actually obtaining appropriate sound and video clips, as they usually require some sort of multimedia capability on behalf of the host machine in order to be able to digitize sound and capture and digitize video. Video suffers from the same problems as graphics, magnified by an order of magnitude or two; it can take extremely large amounts of time for a video segment to download. Video is also not well integrated into the web, requiring the creation of a process to run it that is separate from the page from whence it came. Not all receiving machines have the capability to play video, or sound, and so it is unwise for a designer to rely on these dynamic media to convey information without replicating it elsewhere.

The use of sound and video moves page design further away from the typesetter and toward the sound engineer and cinematographer; the integration of these cinematic media with the enhanced textual capabilities offered by the web is a new domain, in which the techniques that work and those that fail have not yet been fully explored, let alone understood.

The need to download movies and sound (see Figure 21.8) puts sharp limits on the length of clip that can be shown. Streaming media over the internet, such as RealVideo, RealAudio and CuSeeMe, allow potentially unlimited sources. As well as longer prepared clips, these techniques allow live transmission (e.g. live radio broadcasts over RealAudio) and long recorded sequences for asynchronous communication. An excellent use of the latter is the eClass project, introduced in Section 20.2.2, which links recordings of audio and video during a lecture with pen strokes on an electronic whiteboard, so that students can replay the part of a lecture associated with any slide or annotation.

Acceptable streaming video and audio is achieved by a combination of high compression and large client-end buffers. The former leads to loss of quality including blurring and ghosting after rapid changes in screen content. The latter leads to delays, often of several seconds, which makes it impossible to support video conferencing. The challenges of achieving high quality transmissions (e.g. for video on demand) and low latency (e.g. for video conferencing) are active research topics in multimedia technology.

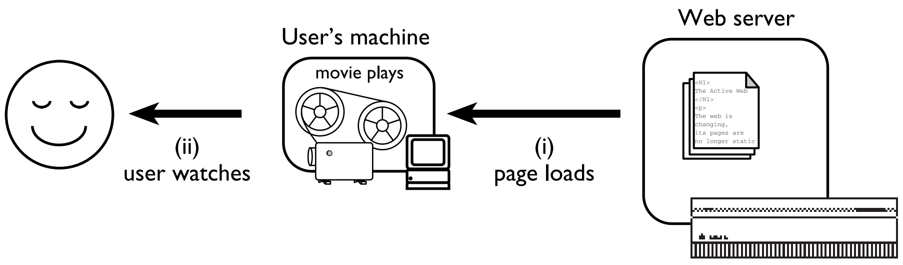


Figure 21.8 Animated GIF or movie needs to download completely

Stepping back a bit to look again at this, note that it is often not raw bandwidth which is the problem on the internet, but packet losses and jitter. Buffering effectively solves this by trading off quality against delay, which is okay for fixed content, or low pace change (as in eClass), but is problematic when we require a *high pace* of interaction (as in video conferencing). CuSeeMe uses little buffering and hence is more likely to suffer break-up of video and audio (see box in Chapter 19, Section 19.3.4, page 675).

21.6 DYNAMIC WEB CONTENT

21.6.1 The active web

In the early days, the web was simply a collection of (largely text) pages linked together. The material was static or slowly changing and much of it authored and updated by hand. Some pages were generated on the fly, in particular the gateways into ftp servers and to gophers, which were so important in adding 'free' content to the web [97]. However, even here the user's model was still of a static repository of information. Web surfers may not have always known where they were, but they had a pretty good idea of what they were seeing and that if they came back it would be the same.

It was a pleasant, if somewhat boring world, but from a usability viewpoint it was wonderful – a consistent interface to terabytes of information. Who could ask for more? Indeed, this is one of the key arguments Nielsen brings against frames-rich sites in his famous alertbox, *Why frames suck (most of the time)* [263] – frames break this simple user model and hence cause trouble. Nielsen calls for a new richer model for the web, which preserves the simplicity of the old model, but which can accommodate and guide the development of new features.

Well, if frames cause trouble, what about applets, timed refreshing pages, rollovers, dynamic content creation? What are we interacting with – is it information, is it computer systems? In fact, this was a problem with hypertext interfaces well before the web existed. Back in 1989, two of the authors, Janet and Alan, wrote about the potential problems of these shifts between passive and active paradigms within an interface. Our solution was to accept these differences, but to make them evident to the user through the design of an effective medium of interaction [103]. Of course it's easy to say . . .

As HCI researchers and designers, we can neither ignore nor uncritically accept new technology in the web. The active web is here, our job is to understand it and to learn how to use it appropriately.

In previous sections, we have already looked at the simplest form of active web page, those with movies, animated gifs or streaming audio. These are simplest, not in the sense that no effort is required – a short video clip may require many days of production effort – but in the sense that they have least user interaction. In this section we'll look at more complex forms of interaction. First, where the actual content is fixed, but the user can change the form of presentation; secondly, at the generation of pages from database content; and finally at the update of database information through the web.

21.6.2 What happens where

When considering dynamic material on the web we need to take the external, user's viewpoint and ask *what* is changing: media, presentation or actual data; by *whom*: by the computer automatically, by the author, by the end-user or by another user; and *how often*, the *pace* of change: seconds, days or months? From a technical standpoint, we also need to know *where* 'computation' is happening: in the user's web-browsing client, in the server, in some other machine or in the human system surrounding it? The 'what happens where' question is the heart of architectural design. It has a major impact on the pace of interaction, both *feedback*, how fast users see the effects of their own actions, and *feedthrough*, how fast they see the effects of others' actions. Also, where the computation happens influences where data has to be moved to with corresponding effects on download times and on the security of the data.

The user view

One set of issues is based on what the end-user sees, the end-user here being the web viewer.

What changes? This may be a media stream (video, audio or animation) which is changing simply because it is the fundamental nature of the medium. It may be the presentation or view the user has of the underlying content; for example, sorting by different categories or choosing text-only views for blind users. A special form of presentation change is when only a selection of the full data set is shown, and that selection changes. The deepest form of change is when the actual content changes.

By whom? Who effects the changes? In the case of a media stream or animation, the changes are largely automatic – made by the computer. The other principal sources of change are the site author and the user. However, in complex sites users may see each other's changes – feedthrough.

How often? Finally, what is the pace of change? Months, days, or while you watch?

We'll use the 'what changes?' categories as we examine alternatives and trade-offs in more detail below. But first we also need to look at the technological constraints.

Technology and security

The fundamental question here is where 'computation' is happening. If pages are changing, there must be some form of 'computation' of those changes. Where does it happen?

Client One answer is in the user's web-browsing client enabled by Java applets, various plug-ins such as Flash, scripting using JavaScript or VBScript with dynamic HTML layers, CSS and DOM (Domain Object Model).

Server A second possibility is at the web server using CGI scripts (written in Perl, C, UNIX shell, Java or whatever you like!), Java Servlets, Active Server Pages or one of the other server-specific scripting languages such as PHP. In addition, client-side Java applets are only allowed to connect to networked resources on the same machine as they came from. This means that databases accessed from client-side JDBC (Java database connectivity) must run on the web server (see below).

Another machine Although the pages are *delivered* from the web server, they may be *constructed* elsewhere. For hand-produced pages, this will usually be on the page author's desktop PC. For generated pages, this may be a PC or a central database server.

People Of course, as noted earlier, the process of production and update may even involve people!

It is easy to roll out maxims such as 'users first', but, in reality, the choice between these options is not solely a matter of matching the end-user requirements. The best choice also depends on the expertise of the web developer and external limitations. If the server runs on a UNIX machine, you can't expect to use Microsoft Active Server Pages. On the other hand, if you are designing for an intranet you may even get to influence the choice of client software and so make it easier to use more complex client-end solutions.

The choice of technological solution is also heavily influenced by issues of security. When we do any computation on data the computation and the data must be in the same place [299]. This apparently simple factor means that if we want to keep data secure (read 'on well-protected servers') then we must also perform the critical computation on the same servers. For example, imagine a password check. It would be foolish to send the correct password to a Java applet to check!

21.6.3 Fixed content – local interaction and changing views

Probably the most hyped aspect of the web in recent years has been Java. In fact, Java can be used to write server-end software and platform independent standalone programs (not to mention the embedded systems for which it was originally designed!), but the aspect that most people think of is Java applets.

Applets are just one of the techniques that can be added to give client-end interaction (and about the least well integrated into the rest of the page). The most common alternatives are JavaScript, Flash and if you are prepared to limit yourself to Windows platforms, ActiveX plug-ins. These techniques share the characteristic that they are downloaded to the user's own machine (see Figure 21.9) and thereafter all interaction happens on the PC, not across the network (with caveats – see below).

The simplest use of this is to add interaction widgets such as roll-over buttons (usually using JavaScript). More complex pages may add the equivalent of an

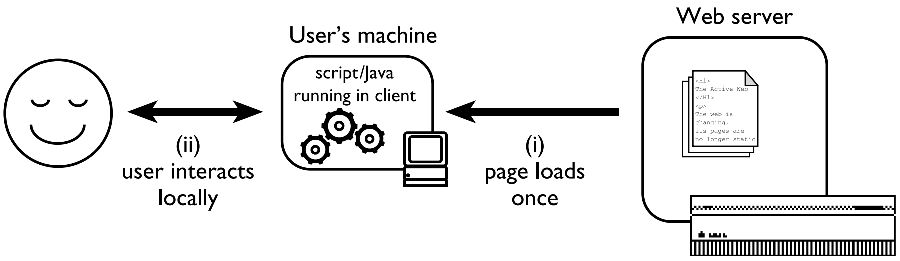


Figure 21.9 Java applet or JavaScript running locally

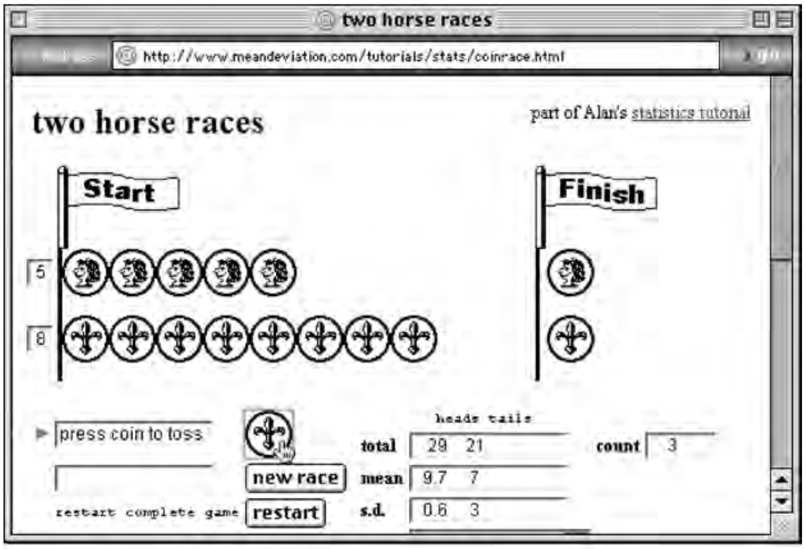
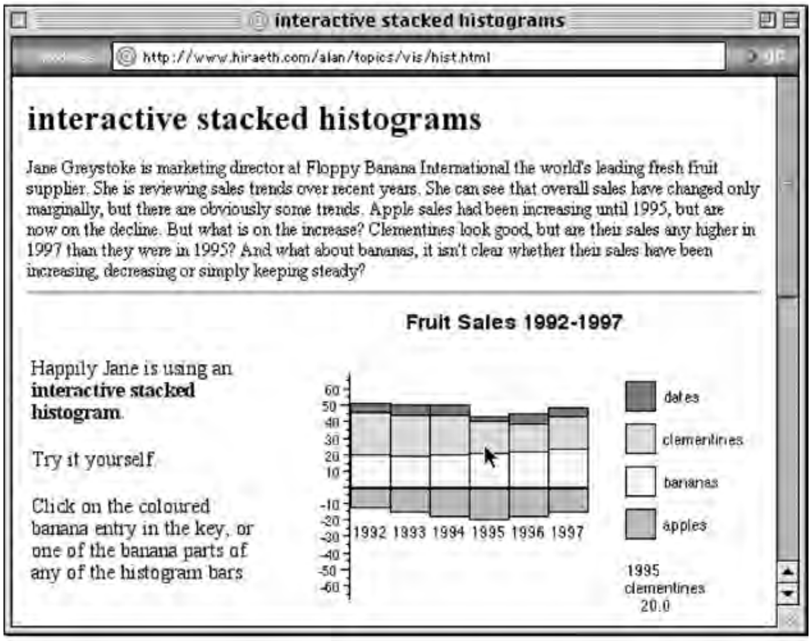


Figure 21.10 Simulated coin tossing using JavaScript. Screen shot frame reprinted by permission from Microsoft Corporation

interactive application on the page. For examples, see Alan's pages on coin tossing experiments (Figure 21.10), which use JavaScript to emulate real and biased coins, and dancing histograms (Figure 21.11), which use a Java applet. See Sun and JavaSoft's own sites for many more examples. The addition of DHTML gives even more opportunities for dynamic pages where parts of the page can move, change size, or change content all without any interaction with the web server.

Notice how this local interaction confuses the static model of the web. What should happen when you go back to a previously visited page, or reload it? Do you get the original state or the last state of your interaction? What happens if you launch a second window on the same page? The actual behavior tends to be browser specific and not always what you would expect! In particular, some browsers do not re-initialize applets on a reload and so if you edit the applet's parameters and then reload you may not see the effects of your changes. More of a problem for web developers than end-users, but very confusing.



interactive stacked histograms

http://www.hiraeth.com/alan/topics/vis/hist.html

interactive stacked histograms

Jane Greystoke is marketing director at Floppy Banana International the world's leading fresh fruit supplier. She is reviewing sales trends over recent years. She can see that overall sales have changed only marginally, but there are obviously some trends. Apple sales had been increasing until 1995, but are now on the decline. But what is on the increase? Clementines look good, but are their sales any higher in 1997 than they were in 1995? And what about bananas, it isn't clear whether their sales have been increasing, decreasing or simply keeping steady?

Happily Jane is using an **interactive stacked histogram**.

Try it yourself.

Click on the coloured banana entry in the key, or one of the banana parts of any of the histogram bars

Figure 21.11 Dancing histograms using Java applet. Screen shot frame reprinted by permission from Microsoft Corporation

21.6.4 Search

Some user-driven interaction can be accommodated at the client end, but not all. Consider search engines. It would be foolish to download several megabytes of information so that a Java applet can search it online! Instead, all common web search pages work by submitting forms to the server where CGI programs perform the searches and return results. An additional reason for this approach is that most browsers support forms, but some still do not support Java or scripting in a consistent manner. The web search engine for this book works in this way. The user's keywords are submitted to the server using an HTML form, they are compared against pre-prepared indexes at the server and all matching paragraphs in the book are returned (Figure 21.12). This also reminds us of another reason for not downloading all the text to the user's machine – security; we don't want to distribute the full electronic text for free!

Notice that, in all the above, the underlying content does not change; the variable factor is the user's input. The computation (working out what to show) needs both the data supplied by the web author (pages, databases, indexes, etc.) and the user's input. The result must end up on the user's screen. Either the data must come to the user's machine (as in Alan's dancing histograms where the histogram data are in applet parameters); or the user's input must go to the server (as with the search). We can see from the examples that the choice between these depends on the required pace of interaction, the size of the data set required, security and available technology.

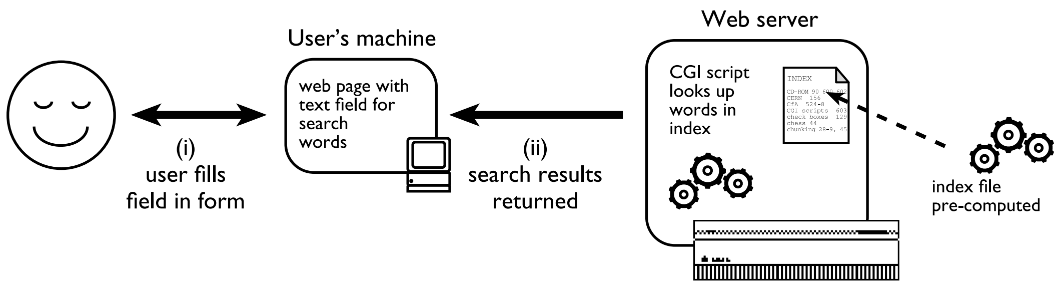

User's machine
web page with text field for search words
(i) user fills field in form
(ii) search results returned
Web server
CGI script looks up words in index
INDEX
index file pre-computed

Figure 21.12 HCI book search

21.6.5 Automatic generation

It was evident in the earliest days of the web that a key problem for the future would be maintenance. In the first rush of enthusiasm, individuals and organizations produced extensive and fascinating sites built largely of hand-crafted HTML. Not surprisingly, vast areas of the web are not just static but in perpetual stasis. Web surfing sometimes seems not so much a watersport, but an exercise in archaeology.

From the beginning it was clear that websites would eventually need to be created from databases of content combined with some form of templates or layout description. However, at that stage there were no tools available and those who saw the database future used a variety of ad hoc methods.

Happily, there are now a (sometimes bewildering) array of products for automating web production from existing and bespoke databases. These include vendor-specific products such as Oracle Web Server and Domino (for publishing Lotus Notes), and also more general techniques such as using SQL (structured query language) or JDBC to access databases from CGI scripts or even from running Java applets.

Database-generated websites have many advantages. They make use of existing data sources. They guarantee consistency of different views of the data within the site and between the site and the corporate data. They allow easy generation of tables of contents, indices, and inter-page links. They separate content and layout.

The most dynamic way to get database content on the web is by accessing a database directly from a running applet. The interface can then have any look-and-feel that can be programmed in Java and can allow very rapid interaction with the user. The Java applet can establish an internet connection back to the web server to access data files using HTTP (as if they were web pages), it can connect to a bespoke server (e.g. for chat type applications) or it can use standard database access methods. The latter would normally use JDBC, the Java database access package. Using JDBC the applet can issue complex SQL queries back to the database meaning that some of the most complex work happens there (Figure 21.13).

In all cases, the Java security model built into most web browsers means that the applet can only connect back to the machine from which it came. This means that the database server must run on the *same* machine as the web server. Think about

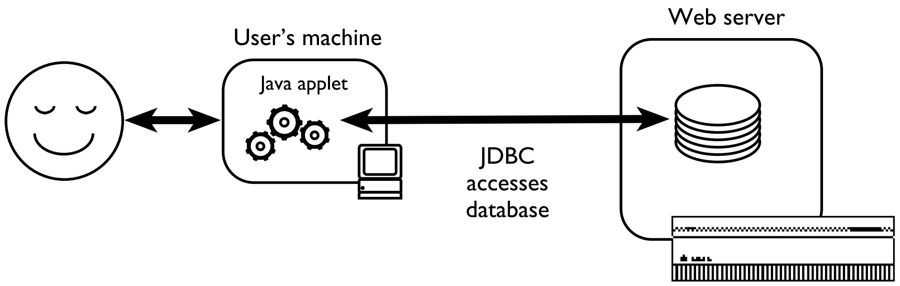


Figure 21.13 Java applet accesses database using JDBC

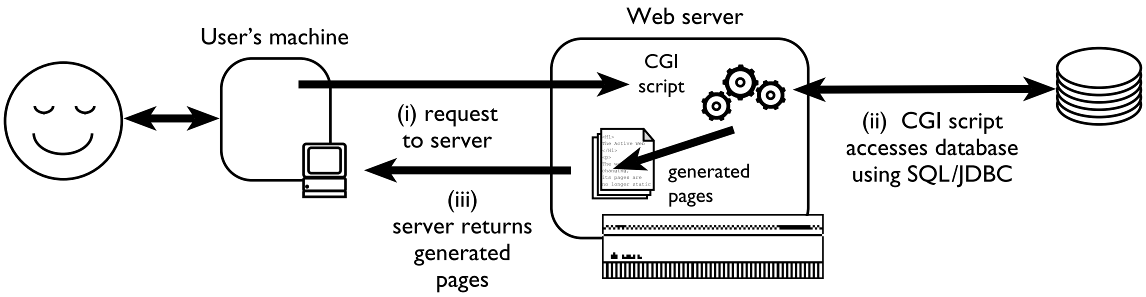


Figure 21.14 CGI script accesses database

this. The most insecure part of any system is usually the web server, both because it is easy to leave loopholes in the many file access permissions and also because it often sits outside the most secure part of a corporate firewall.

The more common solution is where the user uses a web forms interface (or special URL) and then a CGI script runs at the server end accessing the database (Figure 21.14). The CGI script generates a web page, which is then returned to the user. Some of the vendor-specific solutions use essentially this approach but bypass the web-server/CGI step by having their own special web server which accesses the database directly using their own scripting language or templates.

The user interface of such systems is limited to standard HTML features. This is a limitation, but is at least consistent and means that it will work with virtually any browser. Java applets can offer more rapid surface interaction, but both have to wait for the actual data to move between server and client. Of course, the pages generated by a CGI script can themselves contain JavaScript or Java applets for local interaction, so the difference between the two solutions is not so radical as first appears.

From a security angle, the database accessed from the CGI script can run on a separate machine (using standard database remote access methods or even a Java/JDBC CGI program), thus making the system more secure. However, the database cannot be entirely secure – if the web-server machine is compromised the CGI scripts can be altered to corrupt or modify the database! The special vendor-specific web servers are probably more secure as they don't require a standard web server to be running.

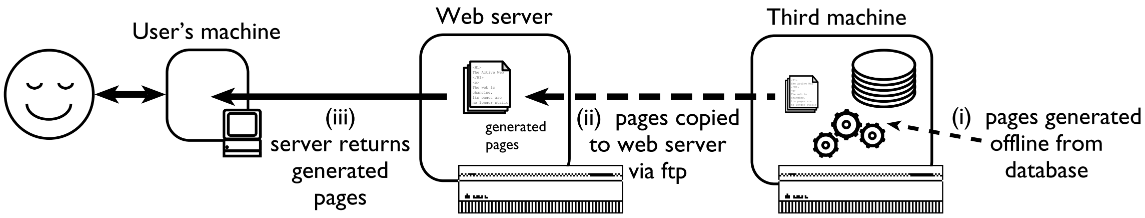

User's machine
Web server
Third machine
(iii) server returns generated pages
generated pages
(ii) pages copied to web server via ftp
(i) pages generated offline from database

Figure 21.15 Batch pre-generation of web pages

21.6.6 Batch generation

A low-tech but very secure solution is to generate pages offline from a database and then upload them to the web server (Figure 21.15). Many of Alan's earliest web pages were generated in this way from HyperCard stacks.

This is certainly a simple solution as it separates out the task of page generation from that of page delivery. Pages can be generated directly using many standard database packages such as Access or HyperCard. Alternatively, standalone programs in languages such as Visual Basic, Java or C can access a database and output HTML pages. These programs can run on a central computer, or on your own PC. The generating program simply produces a set of HTML pages on your own disk that can be checked locally and then copied onto the web server using ftp or shared network disks. Many people think that this will be difficult, but in reality it is remarkably easy, as you can use the tools you are used to – if you can create a text file you can create HTML. In fact, the snippet of Visual Basic in Figure 21.16 is a trivial but fully functioning HTML generator!

```
Set db = openDatabase("C:\test.mdb");
sql = "select Name, Address from
Personnel;"
Set query = db.OpenRecordset(sql)
Open "out.html" For Output As #1

Print #1, "<h1>Address List</h1>"
query.MoveFirst
While Not query.EOF
  Print #1, "<p>" & query("Name") & "  ";
query("Address")
  query.MoveNext
Wend

Close #1
query.Close
```

Figure 21.16 Visual Basic code to generate a web page

Some web scripting languages can be used in this mode too. For example PHP allows you to send the page being generated into a buffer, which can then be saved to a file. This can be run on a separate machine, or on the web server itself. The latter sounds unnecessary; however, web servers are more efficient at delivering static material so this sort of batch generation can be used simply to improve efficiency.

As well as the ease of programming, the offline generation of web pages means that there is no need for an online connection between the web server and the database, so a breach in the security of the web server doesn't compromise your database. In addition, it may mean that the web server can be configured without CGI scripting enabled at all, which considerably increases its security.

The downside is that you can only show the indices and pages that you can precompute. So, you could use a product database to produce a pro-forma page for each stock item, plus alphabetic and categorized lists, but you could not produce a special list based on a user's own search criteria.

This low-tech solution is appropriate in many circumstances – whenever the pace of change is low (e.g. overnight or when periodic updates are acceptable), the volume of data is not too large and no online searching is required. Even when some of these conditions don't hold, it is possible to use the general approach. For example, searching can often be arranged by having a much cut-down database or index on the web server with most pages pre-computed.

21.6.7 Dynamic content

The mechanisms we have been discussing manage the feedthrough when the database is updated by some non-web means. Perhaps the most 'active' web pages are those where the content of the pages reacts to and is updateable by the web user.

If pages are generated from database content using either the Java-applet/JDBC method or the CGI method, the same mechanisms can as easily be used to update as to access the database. The feedback of changes to the user is thus effectively instantaneous – you check for seat availability on the theatre web page, select a seat, enter your credit card details and not only is the seat booked, but you can see it change from free to booked on the web page.

This sort of web application opens up many additional problems. You may need to add some form of login or authentication. If credit card numbers are supplied you need to ensure that the web server is secure. Also, without care it is easy to design solutions that accidentally book multiple seats if the user presses the back button and ends up on what appears to be a simple confirmation screen.

If we consider an estate agent's web page, with houses for sale and potential buyers, the situation is rather different. The pace of change is slow; house purchases take place over days, weeks and months. A solution that automatically marked a house as sold would neither be necessary nor desirable! In this case a socio-technical solution using low-tech database generation would probably be sufficient. The web page can have a contact telephone number, email address or message form. Queries via these channels (as well as non-web-based queries) come to the estate agent who

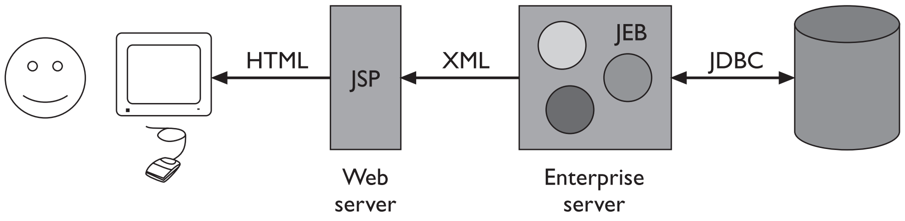


Figure 21.17 *n*-tier architecture

is responsible for deciding when to mark the house 'sold'. There is a continuous loop between web user and the database, but it involves human as well as automatic processes.

Going in the direction of greater complexity, many business applications operate an *n*-tier web architecture. This involves multiple layers of software where the outer layers are concerned more with the user interface and the inner layers more with business functionality. Figure 21.17 shows this using several web standards. The user interacts through a web browser with a web server. The pages are generated using Java Servlet Pages (JSP). To generate the page the servlets connect to Java Enterprise Beans (JEB) on an enterprise server. These are components that encapsulate 'business logic'. For example, in a banking system this could include rules on whether a particular transaction is allowed. These Java Enterprise Beans draw their data from the corporate database using JDBC connections.

21.7 SUMMARY

The non-linear nature of hypertext, in conjunction with the use of different media, can be a powerful combination for education, documentation and marketing. Furthermore, the world wide web has made it possible to publish such material globally at very little cost. However, we saw that users can become confused and disoriented if the structure of a hypertext document is not clear, becoming 'lost in hyperspace'. These problems may become more severe on the web where links cross between sites, and across national borders. Furthermore, web pages are often developed incrementally rather than being pre-planned.

Animation can be used within traditional windowing systems, to aid visualization within special-purpose applications or as part of multimedia systems. The large storage and performance requirements of digital video are still a problem, but one that is gradually being overcome by technological progress. However, the problem of effective use of such technology, beyond the short video clip, may take somewhat longer to resolve.

Dynamic web content can include interactive pages running on the browser or pages generated at the web server from a database. They make use of the same web interface as normal web pages, so in principle they are easy to use. However, they pose additional usability challenges as the user has to understand what is static and what is changing content. Also, it is important to choose carefully where change is happening, whether at the client PC or the web server, as this has a major impact on the nature of interaction, especially the pace of feedback.

The production of multimedia in the form of web pages is now feasible for most computer users and is a good way of making demonstrations and prototypes publicly available. Have a go!

EXERCISES

21.1 Experiment with HyperCard or another hypertext system if you have access to one. As you work through the system, draw a map of the links and connections. Is it clear where you are and where you can get to at any point? If not, how could this be improved?

21.2 Do the same for this book's website and tell us what you think!

21.3 What factors are likely to delay the widespread use of video in interfaces? What applications could benefit most from its use?

21.4 Using a graphics package such as Adobe Photoshop or Macromedia Fireworks save different types of image (photographs, line drawings, text) in different formats (GIF, JPEG, PNG). Compare the file sizes of the different formats, experimenting with different compression ratios (where applicable), numbers of colors, etc.

RECOMMENDED READING

E. J. Conklin, Hypertext: an introduction and survey, *IEEE Computer*, Vol. 20, No. 9, pp. 17–41, September 1987.
A classic review of hypertext.

R. Bentley, U. Busbach, D. Kerr and K. Sikkel, editors, *Groupware and the World Wide Web*, Kluwer, 1997.
The web has become perhaps the most common platform for groupware systems. This collection includes seminal work in this area and covers most of the issues that are still current.

T. Boyle, *Design for Multimedia Learning*, Prentice Hall, 1997.
Covers both educational aspects and multimedia design.

P. Greenspun, *Philip and Alex's Guide to Web Publishing*, Morgan Kaufmann, 1999.
Lovely photos and a strong focus on database-driven web publishing. (Alex probably does his web searching using Lycos.)

J. Niederst, *Learning Web Design: A Beginner's Guide to HTML, Graphics, and Beyond*, O'Reilly, 2001.

CHAPTER 2

The Mobile Ecosystem

The Internet has spoiled us. We tend to oversimplify the technology powering the Internet. The Internet is actually a complex ecosystem made up of many parts that must all work together seamlessly. When you enter a URL into a web browser, you don't think about everything that has to happen to see a web page. When you send an email, you don't care about all the servers, switches, and software that separate you from your recipient. Everything you do on the Internet happens in fractions of a second. And you have the perception that all of this happens for free.

If you talk to people unfamiliar with mobile, you might find that they quickly assume that the mobile ecosystem is exactly like the Internet, and that all the same rules apply. This couldn't be further from the truth. Mobile is an entirely unique ecosystem and, like the Internet, it is made up of many different parts that must all work seamlessly together. However, with mobile technology, the parts are different, and because you can use mobile devices to access the Internet, that means that not only do you need to understand the facets of the Internet, but you also need to understand the mobile ecosystem.

To put it another way, think of the Internet as a great cloud in the sky. When we want to pull something from it, we use a tool, like a piece of software or device, to interact with it. This can include mobile devices, which we tend to think of as tools. Although this image is partially correct, it's still missing a big piece of the puzzle. To continue the analogy, if the Internet is a cloud, then the mobile ecosystem would be the atmosphere, made up of many clouds, keeping the clouds from drifting off into space; the Internet is just one of these clouds, albeit a very large one.

In case that isn't confusing enough, people in mobile tend to refer to *everything* related to mobile as "mobile." This chapter looks at some of the clouds in the sky and how each part plays into the ecosystem as a whole. It also looks at how you can get started with mobile.

Think of the mobile ecosystem instead as a system of layers, as shown in Figure 2-1. Each layer is reliant on the others to create a seamless, end-to-end experience. Although not every piece of the puzzle is included in every mobile product and service, for the

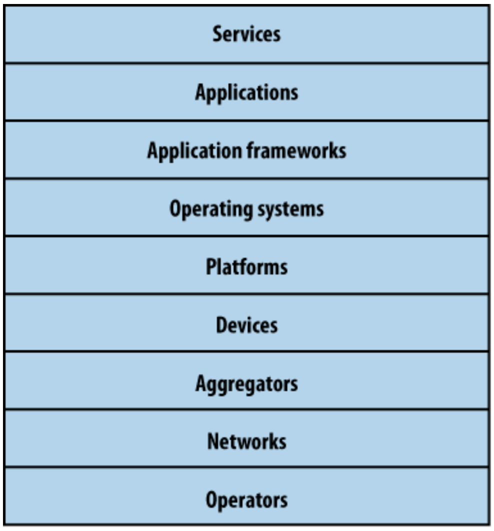


Figure 2-1. The layers of the mobile ecosystem

majority of the time, they seem to add complexity to our work, regardless of whether we expressly put them there.

The following sections expand on each of these layers and the roles they play in the mobile ecosystem.

Operators

The base layer in the mobile ecosystem is the *operator*. Operators go by many names, depending on what part of the world you happen to be in or who you are talking to. Operators can be referred to as Mobile Network Operators (MNOs); mobile service providers, wireless carriers, or simply carriers; mobile phone operators; or cellular companies. In the mobile community, we officially refer to them as operators, though in the United States, there is a tendency to call them *carriers*.

Operators are what essentially make the entire mobile ecosystem work. They are the gatekeepers to the kingdom. They install cellular towers, operate the cellular network, make services (such as the Internet) available for mobile subscribers, and they often maintain relationships with the subscribers, handling billing and support, and offering subsidized device sales and a network of retail stores.

The operator's role in the ecosystem is to create and maintain a specific set of wireless services over a reliable cellular network. That's it. However, to grow the mobile market over the past decade, the operator has been required to take a greater role in the mobile ecosystem, doing much more than just managing the network. For example, they have had to establish trust with subscribers to handle the billing relationship and to offer devices, content, and services that often compete with their partners, who are people like us and who want to create content and services for mobile devices.

Unless you work for an operator, you likely curse their names, at least behind their backs. The operator is viewed as an unfortunate necessity in the mobile world. Often the mobile startups and companies that succeed are the ones with the best "carrier relations man," or person with the best relationship to the operators.

Table 2-1 lists the rank, markets, technologies used, and subscriber numbers for the world's largest operators.

Table 2-1. World's largest mobile operators

Rank	Operator	Markets	Technology	Subscribers (in millions)
1.	China Mobile	China (including Hong Kong) and Pakistan	GSM, GPRS, EDGE, TD-SCDMA	436.12
2.	Vodafone	United Kingdom, Germany, Italy, France, Spain, Romania, Greece, Portugal, Netherlands, Czech Republic, Hungary, Ireland, Albania, Malta, Northern Cyprus, Faroe Islands, India, United States, South Africa, Australia, New Zealand, Turkey, Egypt, Ghana, Fiji, Lesotho, and Mozambique	GSM, GPRS, EDGE, UMTS, HSDPA	260.5
3.	Telefónica	Spain, Argentina, Brazil, Chile, Colombia, Ecuador, El Salvador, Guatemala, Mexico, Nicaragua, Panama, Peru, Uruguay, Venezuela, Ireland, Germany, United Kingdom, Czech Republic, Morocco, and Slovakia	CDMA, CDMA2000 1x, EV-DO, GSM, GPRS, EDGE, UMTS, HSDPA	188.9
4.	América Móvil	United States, Argentina, Chile, Colombia, Paraguay, Uruguay, Mexico, Puerto Rico, Ecuador, Jamaica, Peru, Brazil, Dominican Republic, Guatemala, Honduras, Nicaragua, Ecuador, and El Salvador	CDMA, CDMA2000 1x, EV-DO, GSM, GPRS, EDGE, UMTS, HSDPA	172.5
5.	Telenor	Norway, Sweden, Denmark, Hungary, Montenegro, Serbia, Russia, Ukraine, Thailand, Bangladesh, Pakistan, and Malaysia	GSM, GPRS, EDGE, UMTS, HSDPA	143.0
6.	China Unicom	China	GSM, GPRS	127.6
7.	T-Mobile	Germany, United States, United Kingdom, Poland, Czech Republic, Netherlands, Hungary, Austria, Croatia, Slovakia, Macedonia, Montenegro, Puerto Rico, and U.S. Virgin Islands	GSM, GPRS, EDGE, UMTS, HSDPA	126.6
8.	TeliaSonera	Norway, Sweden, Denmark, Finland, Estonia, Latvia, Lithuania, Spain, and Central Asia	GSM, GPRS, EDGE, UMTS, HSDPA	115.0

Rank	Operator	Markets	Technology	Subscribers (in millions)
9.	Orange	France, United Kingdom, Switzerland, Poland, Spain, Romania, Moldova, Slovakia, Belgium, Liechtenstein, Israel, Egypt, Ivory Coast, Jordan, Cameroon, Botswana, Madagascar, Mali, Senegal, Mauritius, Réunion, Martinique, French Guiana, Saint Kitts and Nevis, Dominica, and Dominican Republic	GSM, GPRS, EDGE, UMTS, HSDPA	111.8
10.	MTS	Russia, Ukraine, Belarus, Uzbekistan, Turkmenistan, and Armenia	GSM, GPRS, EDGE, UMTS	91.7
11.	MTN Group	Afghanistan, Benin, Botswana, Cameroon, Republic of Congo, Côte d'Ivoire, Cyprus, Ghana, Guinea Bissau, Republic of Guinea, Iran, Liberia, Nigeria, Rwanda, South Africa, Sudan, Swaziland, Syria, Uganda, Yemen, and Zambia	GSM, GPRS, EDGE, UMTS, HSDPA, HSUPA	80.7
12.	AT&T	United States, Puerto Rico, and U.S. Virgin Islands	GSM, GPRS, EDGE, UMTS, HSDPA	74.9
13.	Bharti Airtel	India, Seychelles, Jersey, Guernsey, and Sri Lanka	GSM, GPRS, EDGE	72.0
14.	Verizon Wireless	United States	CDMA2000 1x, EV-DO	70.8
15.	SingTel	Singapore, Australia, India, Indonesia, Thailand, Philippines, Bangladesh, and Pakistan	GSM, UMTS, HSDPA	70.7
16.	Telecom	Italy, Brazil, San Marino, and Vatican City	GSM, GPRS, EDGE, UMTS, HSDPA	70.6
17.	Etisalat	Afghanistan, Benin, Burkina Faso, Central African Republic, the Ivory Coast, Egypt, Gabon, Indonesia, Niger, Nigeria, Pakistan, Saudi Arabia, Sudan, Tanzania, Togo, and United Arab Emirates	GSM, GPRS, EDGE, UMTS, HSDPA	63.0
18.	Orascom	Algeria, Bangladesh, Egypt, Pakistan, Tunisia, and Zimbabwe	GSM, GPRS, EDGE	62.9
19.	VimpelCom	Russia, Kazakhstan, Ukraine, Uzbekistan, Tajikistan, Georgia, Armenia, Vietnam, and Cambodia	GSM, GPRS, UMTS	57.8
20.	NTT docomo	Japan and Bangladesh	GSM, GPRS, PDC FOMA, HSDPA	53.5

Although most operators are interested in innovation in the wireless marketplace, they have been known to strangle startups with impossible requirements, such as supporting too many devices or seemingly ridiculous certification processes and bad pricing models.

The days of impossible requirements are changing, however. Today's mobile startups have learned the lessons of the companies that came before them that tried to dance with the devil and lost everything. Many look to the successes of Web 2.0–era startups that were able to start with little infrastructure and quickly grow successful businesses. These startups figured out how to duplicate the phenomenon of mobile, bypassing the operators completely (something this book tells you how to do).

You can compare operators to Big Oil. They both have this thing they know everyone wants, and therefore they can make a lot of money from it. They know they have a limited amount of time to make it. With oil, it is the depleted resources and competition of green energy sources; in wireless, it's the growth of competing wireless technologies, such as Wi-Fi, WiMAX, ultra-wide broadband, and whitespace frequencies.

As competing technologies become more mature, they can't charge as much as they did when they first came out. As consumer options in the market mature, both the oil industry and operators must realize that they can't continue to monopolize their markets. They must realize that they don't control their industries; they are only a player in them. Unfortunately, in the meantime, both of these industries will continue to force us to pay an artificially inflated cost to play.

Networks

Operators operate wireless networks. Remember that cellular technology is just a radio that receives a signal from an antenna. The type of radio and antenna determines the capability of the network and the services you can enable on it.

You'll notice that the vast majority of networks around the world use the GSM standard (see Table 2-2 for an explanation of these acronyms), using GPRS or GPRS EDGE for 2G data and UMTS or HSDPA for 3G. We also have CDMA (Code Division Multiple Access) and its 2.5G hybrid CDMA2000, which offers greater coverage than its more widely adopted rival. So in places like the United States or China, where people are more spread out, CDMA is a great technology. It uses fewer towers, giving subscribers fewer options as they roam networks.

Table 2-2. GSM mobile network evolutions

2G	Second generation of mobile phone standards and technology	Theoretical max data speed
GSM	Global System for Mobile communications	12.2 KB/sec
GPRS	General Packet Radio Service	Max 60 KB/sec
EDGE	Enhanced Data rates for GSM Evolution	59.2 KB/sec
HSCSD	High-Speed Circuit-Switched Data	57.6 KB/sec

3G	Third generation of mobile phone standards and technology	Theoretical max data speed
W-CDMA	Wideband Code Division Multiple Access	14.4 MB/sec
UMTS	Universal Mobile Telecommunications System	3.6 MB/sec
UMTS-TDD	UMTS +Time Division Duplexing	16 MB/sec
TD-CDMA	Time Divided Code Division Multiple Access	16 MB/sec
HSPA	High-Speed Packet Access	14.4 MB/sec
HSDPA	High-Speed Downlink Packet Access	14.4 MB/sec
HSUPA	High-Speed Uplink Packet Access	5.76 MB/sec

Like all things in mobile, we like to merge a lot of technology into overly simplistic terms, which tends to create a lot of confusion. So when we say 3G, for example, we often aren't talking about just the capabilities of the network, but the devices that run on it.

Although the core technology that empowers voice communication has stayed relatively the same, network generations are most often used to describe the data speeds the network is capable of delivering.

Devices

What you call phones, the mobile industry calls handsets or terminals. These are terms that I think are becoming outdated with the emergence of wireless devices that rely on operator networks, but do not make phone calls. The number of these "other" devices is a small piece of the overall pie right now, but it's growing rapidly.

Let's focus on the biggest slice of the device pie—mobile phones. As of 2008, there are about 3.6 billion mobile phones currently in use around the world; just more than half the planet's population has a mobile phone (see Figure 2-2).

Most of these devices are feature phones, making up the majority of the marketplace. Smartphones make up a small sliver of worldwide market share and maintain a healthy percentage in the United States and the European Union; smartphone market share is growing with the introduction of the iPhone and devices based on the Android platform. As next-generation devices become a reality, the distinction between feature phones and smartphones will go away. In the next few years, feature phones will largely be located in emerging and developing markets. Figure 2-3 shows a breakdown of devices.

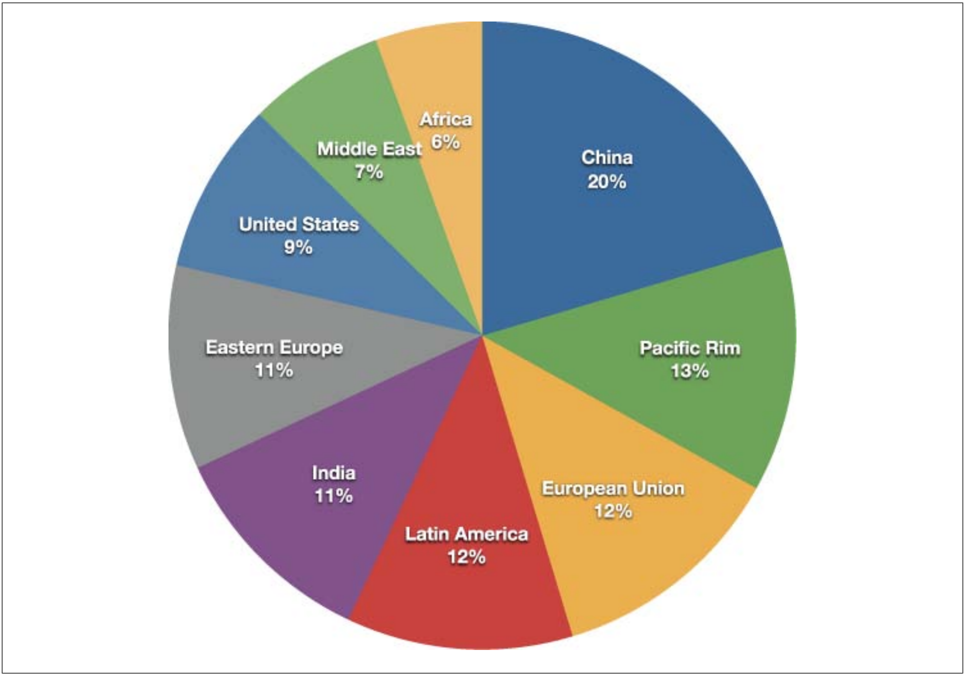


Figure 2-2. Mobile devices around the world

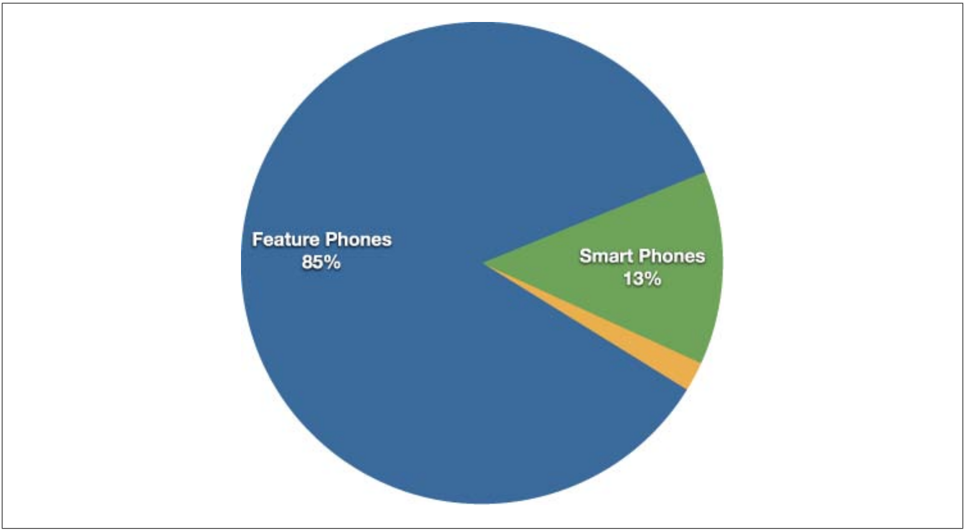


Figure 2-3. Breakdown of devices

Most mobile devices are subsidized in some form or another. Operators sell devices at a severely discounted price, often one-third or less of the actual cost of the device. This enables the operators to lock the devices to their networks. They can then preload onto the device content and services that are beneficial to themselves in exchange for lower price points, encouraging subscribers to upgrade to new devices with new capabilities.

Subsidization means that devices need to be *provisioned* (or customized) to operators' individual requirements. Provisioning dramatically increases the number of devices released every year, with each device being slightly different from the other.

The sheer number of devices is both a blessing and a curse to the mobile industry. On the one hand, the magnitude of the mobile market is huge. It is one of the largest digital mediums mankind has ever seen. On the other hand, so many devices means adapting to those devices—not to mention painful and costly development cycles.

This brings us to the greatest challenge the mobile ecosystem currently faces: device fragmentation, a term used to describe how mobile devices interpret industry specifications differently, causing different mobile devices to display content inconsistently. Despite what you may know or have heard, you can take a deep breath and relax. Device fragmentation is a topic we will clear up completely in the following chapters.

Platforms

A mobile platform's primary duty is to provide access to the devices. To run software and services on each of these devices, you need a *platform*, or a core programming language in which all of your software is written. Like all software platforms, these are split into three categories: licensed, proprietary, and open source.

Licensed

Licensed platforms are sold to device makers for nonexclusive distribution on devices. The goal is to create a common platform of development Application Programming Interfaces (APIs) that work similarly across multiple devices with the least possible effort required to adapt for device differences, although this is hardly reality. Following are the licensed platforms:

Java Micro Edition (Java ME)
: Formerly known as J2ME, Java ME is by far the most predominant software platform of any kind in the mobile ecosystem. It is a licensed subset of the Java platform and provides a collection of Java APIs for the development of software for resource-constrained devices such as phones.

Binary Runtime Environment for Wireless (BREW)
: BREW is a licensed platform created by Qualcomm for mobile devices, mostly for the U.S. market. It is an interface-independent platform that runs a variety of application frameworks, such as C/C++, Java, and Flash Lite.

Windows Mobile
: Windows Mobile is a licensable and compact version of the Windows operating system, combined with a suite of basic applications for mobile devices that is based on the Microsoft Win32 API.

LiMo
: LiMo is a Linux-based mobile platform created by the LiMo Foundation. Although Linux is open source, LiMo is a licensed mobile platform used for mobile devices. LiMo includes SDKs for creating Java, native, or mobile web applications using the WebKit browser framework.

Proprietary

Proprietary platforms are designed and developed by device makers for use on their devices. They are not available for use by competing device makers. These include:

Palm
: Palm uses three different proprietary platforms. Their first and most recognizable is the Palm OS platform based on the C/C++ programming language; this was initially developed for their Palm Pilot line, but is now used in low-end smartphones such as the Centro line. As Palm moved into higher-end smartphones, they started using the Windows Mobile-based platform for devices like the Treo line. The most recent platform is called webOS, is based on the WebKit browser framework, and is used in the Prē line.

BlackBerry
: Research in Motion maintains their own proprietary Java-based platform, used exclusively by their BlackBerry devices.

iPhone
: Apple uses a proprietary version of Mac OS X as a platform for their iPhone and iPod touch line of devices, which is based on Unix.

Open Source

Open source platforms are mobile platforms that are freely available for users to download, alter, and edit. Open source mobile platforms are newer and slightly controversial, but they are increasingly gaining traction with device makers and developers. Android is one of these platforms. It is developed by the Open Handset Alliance, which is spearheaded by Google. The Alliance seeks to develop an open source mobile platform based on the Java programming language.

Operating Systems

It used to be that if a mobile device ran an operating system, it was most likely considered a smartphone. But as technology gets smaller, a broader set of devices supports operating systems.

Operating systems often have core services or toolkits that enable applications to talk to each other and share data or services. Mobile devices without operating systems typically run "walled" applications that do not talk to anything else.

Although not all phones have operating systems, the following are some of the most common:

Symbian
: Symbian OS is a open source operating system designed for mobile devices, with associated libraries, user interface frameworks, and reference implementations of common tools.

Windows Mobile
: Windows Mobile is the mobile operating system that runs on top of the Windows Mobile platform.

Palm OS
: Palm OS is the operating system used in Palm's lower-end Centro line of mobile phones.

Linux
: The open source Linux is being increasingly used as an operating system to power smartphones, including Motorola's RAZR2.

Mac OS X
: A specialized version of Mac OS X is the operating system used in Apple's iPhone and iPod touch.

Android
: Android runs its own open source operating system, which can be customized by operators and device manufacturers.

You might notice that many of these operating systems share the same names as the platforms on which they run. Mobile operating systems are often bundled with the platform they are designed to run on.

Application Frameworks

Often, the first layer the developer can access is the application framework or API released by one of the companies mentioned already. The first layer that you have any control over is the choice of application framework.

Application frameworks often run on top of operating systems, sharing core services such as communications, messaging, graphics, location, security, authentication, and many others.

Java

Applications written in the Java ME framework can often be deployed across the majority of Java-based devices, but given the diversity of device screen size and processor power, cross-device deployment can be a challenge.

Most Java applications are purchased and distributed through the operator, but they can also be downloaded and installed via cable or over the air.

S60

The S60 platform, formerly known as Series 60, is the application platform for devices that run the Symbian OS. S60 is often associated with Nokia devices—Nokia owns the platform—but it also runs on several non-Nokia devices. S60 is an open source framework.

S60 applications can be created in Java, the Symbian C++ framework, or even Flash Lite.

BREW

Applications written in the BREW application framework can be deployed across the majority of BREW-based devices, with slightly less cross-device adaption than other frameworks.

However BREW applications must go through a costly and timely certification process and can be distributed only through an operator.

Flash Lite

Adobe Flash Lite is an application framework that uses the Flash Lite and ActionScript frameworks to create vector-based applications. Flash Lite applications can be run within the Flash Lite Player, which is available in a handful of devices around the world.

Flash Lite is a promising and powerful platform, but there has been some difficulty getting it on devices. A distribution service for applications written in Flash Lite is long overdue.

Windows Mobile

Applications written using the Win32 API can be deployed across the majority of Windows Mobile-based devices. Like Java, Windows Mobile applications can be downloaded and installed over the air or loaded via a cable-connected computer.

Cocoa Touch

Cocoa Touch is the API used to create native applications for the iPhone and iPod touch. Cocoa Touch applications must be submitted and certified by Apple before being included in the App Store. Once in the App Store, applications can be purchased, downloaded, and installed over the air or via a cable-connected computer.

Android SDK

The Android SDK allows developers to create native applications for any device that runs the Android platform. By using the Android SDK, developers can write applications in C/C++ or use a Java virtual machine included in the OS that allows the creation of applications with Java, which is more common in the mobile ecosystem.

Web Runtimes (WRTs)

Nokia, Opera, and Yahoo! provide various Web Runtimes, or WRTs. These are meant to be miniframeworks, based on web standards, to create mobile widgets. Both Opera's and Nokia's WRTs meet the W3C-recommended specifications for mobile widgets.

Although WRTs are very interesting and provide access to some device functions using mobile web principles, I've found them to be more complex than just creating a simple mobile web app, as they force the developer to code within an SDK rather than just code a simple web app. And based on the number of mobile web apps written for the iPhone versus the number written for other, more full-featured WRTs, I don't think I'm alone in thinking this. Nonetheless, it is a move in the right direction.

WebKit

With Palm's introduction of webOS, a mobile platform based on WebKit, and given its predominance as a mobile browser included in mobile platforms like the iPhone, Android, and S60, and that the vast majority of mobile web apps are written specifically for WebKit, I believe we can now refer to WebKit as a mobile framework in its own right.

WebKit is a browser technology, so applications can be created simply by using web technologies such as HTML, CSS, and JavaScript. WebKit also supports a number of recommended standards not yet implemented in many desktop browsers.

Applications can be run and tested in any WebKit browser, desktop, or mobile device.

The Web

The Web is the only application framework that works across virtually all devices and all platforms. Although innovation and usage of the Web as an application framework in mobile has been lacking for many years, increased demand to offer products and services outside of operator control, together with a desire to support more devices in shorter development cycles, has made the Web one of the most rapidly growing mobile application platforms to date.

Applications

Application frameworks are used to create applications, such as a game, a web browser, a camera, or media player. Although the frameworks are well standardized, the devices are not. The largest challenge of deploying applications is knowing the specific device attributes and capabilities. For example, if you are creating an application using the Java ME application framework, you need to know what version of Java ME the device supports, the screen dimensions, the processor power, the graphics capabilities, the number of buttons it has, and how the buttons are oriented. Multiply that by just a few additional handsets and you have hundreds of variables to consider when building an application. Multiply it by the most popular handsets in a single market and you can easily have a thousand variables, quickly dooming your application's design or development.

Although mobile applications can typically provide an excellent user experience, it almost always comes at a fantastic development cost, making it nearly impossible to create a scalable product that could potentially create a positive return on investment.

A common alternative these days is creating applications for only one platform, such as the iPhone or Android. By minimizing the number of platforms the developer has to support and utilizing modern application frameworks, the time and cost of creation go down significantly. This strategy may be perfectly acceptable to many, but what about the rest of the market? Surely people without a more costly smartphone should be able to benefit from mobile applications, too.

Many see the web browser as the solution to this problem and the savior from the insanity of deploying multidevice applications. The mobile web browser is an application that renders content that is device-, platform-, and operating-system-independent. The web browser knows its limitations, enabling content to scale gracefully across multiple screen sizes. However, like all applications, mobile web browsers suffer from many of the same device fragmentation problems.

You could consider the Motorola RAZR to be the epitome of the mobile ecosystem of yesterday. It's been provisioned to numerous operators around the world. It's the perfect example not just of how crazy deploying mobile applications to devices can be, but also of just how bad mobile web browser fragmentation can be. It is a highly prolific device and one that is often recommended for people to support, due to its market penetration. But that is much easier said than done.

If you look at the WURFL database (an open source device repository that is discussed later in this book), you can see that the V3, the real name of the RAZR, has an Openwave 6.2.3.2 web browser. The V3/I/R had the Openwave 6.2.3.4.C.1.109 browser; the V3M/V9M had the Teleca Obigo 4.0 browser; the V3X had the Openwave 6.2.3.1.C. 1.112 browser; the V3M had the Openwave 6.2.3.1.C.1.115 browser; the V3XXI had the Opera 8.0 browser; and the V8 had the Opera 8.5 browser. This isn't even half the list!

From the consumer and business perspective, these are all Motorola RAZRs. But in terms of supporting the RAZR, these might as well be seven different devices. Each of these RAZRs carries very different versions of common applications, each customized for the operator on which they are intended to work.

When a device is sold to an operator, it is provisioned (customized) to their requirements. This means the operators will often put customized applications on each of the devices sold. With the example of the RAZR, every operator had it and every operator put a different web browser on it. To make matters worse, the RAZRs, like most phones, are not field-refreshable, meaning that you can't update the software, upgrade the applications, or eliminate bugs.

For example, if a device manufacturer makes a device called the MDv1, they must strike a deal with an operator if they want to preload an operator store application, a different web browser, and bowling game. The device is sold as the MDv1.1. The operator sells the devices, or worse, gives them away for free. A couple hundred thousand of them go out into the marketplace before a glitch in the hardware is detected, such as dropped calls. Because the device cannot be upgraded by cable or over the air, the operator stops selling the MDv1.1, but seeing that they have a hit, they quickly replace it with the MDv1.1.1. The whole process is repeated as it is provisioned to each operator. Suddenly, there is an MDv1.2, an MDv1.3, an MDv1.4, and so on. Then we have the next generations—the MDv1.2.1, the MDv1.3.1, the MDv1.4.1, and so on, spreading like a virus. This is essentially what causes device fragmentation, making application development a costly and timely endeavor.

Services

Finally, we come to the last layer in the mobile ecosystem: services. Services include tasks such as accessing the Internet, sending a text message, or being able to get a location—basically, anything the user is trying to do.

What makes the mobile environment such a complicated space to design and develop for are these layers, which the user must wade through in order to accomplish a simple task like "I want to send a text message," "I want to get on the Web," and "I want to access Google." The user has so many opportunities for failure that creating a valuable experience is virtually impossible.

How do everyday people use their mobile devices? What are their impressions of the mobile web? Here is some of the consistent feedback you might hear if you ask these questions:

> "It's crap."
>
> "I don't use the mobile web, just because it's awful."
>
> "It costs too much."
>
> "I don't know where my browser is."
>
> "I don't know how to enter the URL."
>
> "I want to go to Wikipedia, but I don't know what to do."
>
> "How do I check my email?"

That is, of course, only if you are old—past your early 30s. The younger you are, the more likely you are rely on mobile services for daily information. Earlier generations—those born since the birth of the Internet—have a unique talent for being able to figure out complicated informational spaces. They are more patient with technology and more apt to explore new methods of accomplishing tasks.

And although one day the youth of today will inherit the digital world, for the time being, the mobile ecosystem is a complicated, fragmented, political nightmare. If I were an entrepreneur looking to create mobile services, knowing what I know about mobile, I would run away, fast. I would probably open a restaurant, which would likely have a higher chance of success.

But we've already seen the future of mobile development, in the form of the iPhone. The iPhone attempts to solve many of the problems facing the mobile ecosystem, from how people interact with their phones, to where we buy our phones, to what type of applications we will pay money for, to the level of technology standards we can support on constrained devices. What makes the iPhone special is how it attempts change on virtually all fronts, something no other device, or company for that matter, has been able to do previously.

Now Apple has done it, the gates are wide open for anyone. People in the industry aren't as jaded anymore, and there is a feeling of excitement and optimism. Although many of the problems in the mobile ecosystem are yet to be fixed and we still have plenty of nonsense to contend with, we can see the light. We can see the path to innovation, to creating applications and services that can quite literally reach the entire planet and quite possibly change the world.

It begins here, with you, right now. Today is the first day of the future of mobile.

CHAPTER 6

Types of Mobile Applications

As we've already discussed, the mobile ecosystem is a large and deep pool. In fact, it probably isn't a pool, but an ocean—a really, really big ocean. An ocean is a good metaphor to put the different types of mobile applications in context. You see, in order to traverse an ocean, you need a sturdy boat. Boats of course come in all sorts of shapes and sizes, each with their pros and cons. Some are fast and agile, but carry little cargo. Others are large and lumbering, but can carry tons of people or cargo.

Mobile applications aren't that much different from boats in this seafaring example. You have a number of choices in what medium you use to address your goals, each with their own pros and cons. Some are quick to create but accessible to fewer users. Others address a larger market, but are far more complex and costly.

Alas, deciding what medium type (or types in some cases) to use gets only you halfway to your destination. You have to decide in what type of application context you want to present your content or information. In other words, what type of application is best suited to your problem or need? Should your app be focused on presenting information? If so, how concisely should you present it? Or is your app better suited to be an immersive experience? If yes, how immersive should it be, and is it a widget or a game? You might think it sounds silly to mention making a game, but for the mobile user, a game can be a great way to get your point across.

Although I'm a big proponent of looking at the mobile web first in the majority of cases, I find that each time I talk to a new company that wants to start with a mobile strategy, the company is simply confused about where to start. There isn't an understanding of the differences between the mobile web and a downloadable application, not to mention the pros and cons of each. Even within the mobile community, you can't tell the difference between a mobile web application and a mobile widget.

This chapter discusses the various types of mobile applications and defines each of these options. First, mobile options and their pros and cons are discussed. Those options are then discussed in an application context—how the application is presented to the user and how to leverage it effectively.

Mobile Application Medium Types

The *mobile medium type* is the type of application framework or mobile technology that presents content or information to the user. It is a technical approach regarding which type of medium to use; this decision is determined by the impact it will have on the user experience. The technical capabilities and capacity of the publisher also factor into which approach to take.

Earlier I discussed the common mobile platforms in terms of how they factor in the larger mobile ecosystem. Now we will look deeper into each of these platforms from a more tactical perspective, unpacking them, so to speak, to see what is inside.

Figure 6-1 illustrates the spectrum of mobile media; it starts with the basic text-based experiences and moves on to the more immersive experiences.

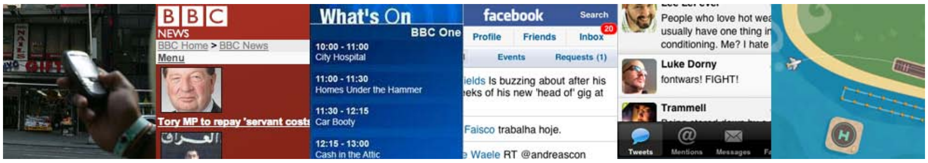

BBC NEWS
BBC Home > BBC News
Menu
Tory MP to repay 'servant costs'
What's On
BBC One
10:00 - 11:00 City Hospital
11:00 - 11:30 Homes Under the Hammer
11:30 - 12:15 Car Booty
12:15 - 13:00 Cash in the Attic
facebook
Search
Profile
Friends
Inbox
Events
Requests (1)
People who love hot weather usually have one thing in common: conditioning. Me? I hate
Luke Dorny
fontwars! FIGHT!
Trammell
Tweets
Mentions
Messages

Figure 6-1. Multiple mobile application medium types

SMS

The most basic mobile application you can create is an SMS application. Although it might seem odd to consider text messages applications, they are nonetheless a designed experience. Given the ubiquity of devices that support SMS, these applications can be useful tools when integrated with other mobile application types.

Typically, the user sends a single keyword to a five-digit short code in order to return information or a link to premium content. For example, sending the keyword “freebie” to a hypothetical short code “12345” might return a text message with a coupon code that could be redeemed at a retail location, or it could include a link to a free ringtone.

SMS applications can be both “free,” meaning that there is no additional charge beyond the text message fees an operator charges, or “premium,” meaning that you are charged an additional fee in exchange for access to premium content.

The most common uses of SMS applications are mobile content, such ringtones and images, and to interact with actual goods and services. Some vending machines can dispense beverages when you send them an SMS; SMS messages can also be used to purchase time at a parking meter or pay lot.

A great example of how SMS adds incredible value would be Twitter, where users can receive SMS alerts from their friends and post to their timeline from any mobile device, or the SMS-to-Billboard that BBC World News put up in Midtown Manhattan (Figure 6-2).

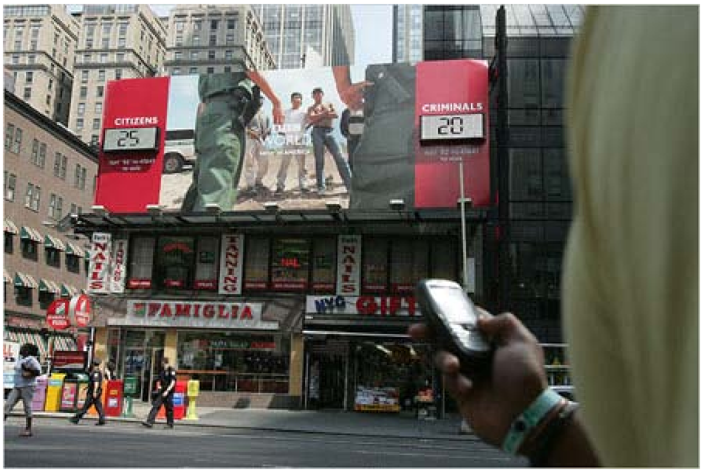


Figure 6-2. An SMS application to interact with a billboard in Manhattan

Pros

The pros of SMS applications include:

- They work on any mobile device nearly instantaneously.
- They're useful for sending timely alerts to the user.
- They can be incorporated into any web or mobile application.
- They can be simple to set up and manage.

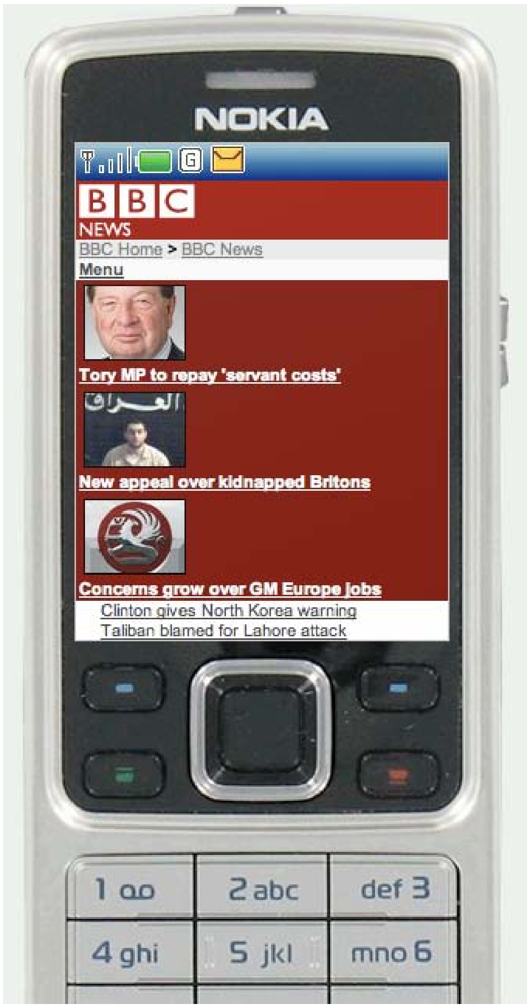
Cons

The cons of SMS applications include:

- They're limited to 160 characters.
- They provide a limited text-based experience.
- They can be very expensive.

Mobile Websites

As you might expect, a mobile website is a website designed specifically for mobile devices, not to be confused with viewing a site made for desktop browsers on a mobile browser. Mobile websites are characterized by their simple "drill-down" architecture, or the simple presentation of navigation links that take you to a page a level deeper, as shown in Figure 6-3.



BBC Home > BBC News
Menu
Tory MP to repay 'servant costs'
New appeal over kidnapped Britons
Concerns grow over GM Europe jobs
Clinton gives North Korea warning
Taliban blamed for Lahore attack

Figure 6-3. An example of a mobile website

Mobile websites often have a simple design and are typically informational in nature, offering few—if any—of the interactive elements you might expect from a desktop site. Mobile websites have made up the majority of what we consider the mobile web for the past decade, starting with the early WML-based sites (not much more than a list of links) and moving to today's websites, with a richer experience that more closely resembles the visual aesthetic users have come to expect with web content.

Though mobile websites are fairly easy to create, they fail to display consistently across multiple mobile browsers—a trait common to all mobile web mediums. The mobile web has been gradually increasing in usage over the years in most major markets, but the limited experience offered little incentive to the user. Many compare the mobile web to a 10-year-old version of the Web: slow, expensive to use, and not much to look at.

As better mobile browsers started being introduced to device platforms like the iPhone and Android, the quality of mobile websites began to improve dramatically, and with it, usage improved. For example, in just one year, the U.S. market went from being just barely in the top five consumers of the mobile web to number one, largely due to the impact of the iPhone alone.

Pros

The pros of mobile websites are:

- They are easy to create, maintain, and publish.
- They can use all the same tools and techniques you might already use for desktop sites.
- Nearly all mobile devices can view mobile websites.

Cons

The cons of mobile websites are:

- They can be difficult to support across multiple devices.
- They offer users a limited experience.
- Most mobile websites are simply desktop content reformatted for mobile devices.
- They can load pages slowly, due to network latency.

Mobile Web Widgets

Largely in response to the poor experience provided by the mobile web over the years, there has been a growing movement to establish mobile widget frameworks and platforms. For years the mobile web user experience was severely underutilized and failed to gain traction in the market, so several operators, device makers, and publishers began creating widget platforms (Figure 6-4) to counter the mobile web's weaknesses.

Trying to define what exactly a mobile web widget is and how it is different from the other mobile web media is a question for the ages. I initially saw mobile web widgets as another attempt by the mobile industry to hype a technology that no one wants. I liked to quiz mobile web widget advocates about what makes mobile web widgets different than what we can do with the mobile web. I was never able to get a straight answer. So in order to define a mobile web widget, I followed some advice from my dad: "When in doubt, look it up in the dictionary." Here was the answer I found in *Webster's Dictionary*:

> A component of a user interface that operates in a particular way.

The ever-trusty Wikipedia defines a web widget this way:

> A portable chunk of code that can be installed and executed within any separate HTML-based web page by an end user without requiring additional compilation.

Between these two definitions is a better answer:

> A mobile web widget is a standalone chunk of HTML-based code that is executed by the end user in a particular way.

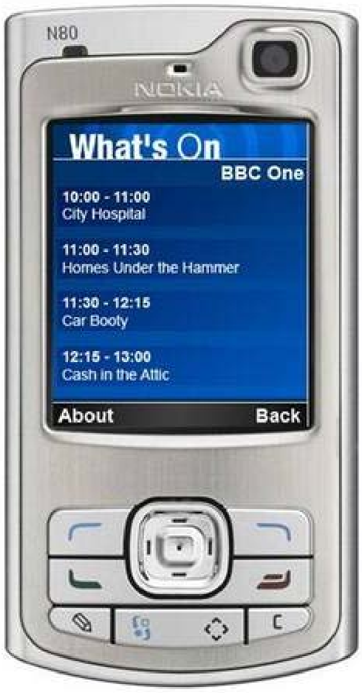


Figure 6-4. An example mobile web widget

Basically, mobile web widgets are small web applications that can't run by themselves; they need to be executed on top of something else. I think one reason for all the confusion around what is a mobile web widget is that this definition can also encompass any web application that runs in a browser. Opera Widgets, Nokia Web RunTime (WRT), Yahoo! Blueprint, and Adobe Flash Lite are all examples of widget platforms that work on a number of mobile handsets. Using a basic knowledge of HTML (or vector graphics in the case of Flash), you can create compelling user experiences that tap into device features and, in many cases, can run while the device is offline.

Widgets, however, are not to be confused with the utility application context, a user experience designed around short, task-based operations, which I will explain in depth later in this chapter.

Pros

The pros of mobile web widgets are:

- They are easy to create, using basic HTML, CSS, and JavaScript knowledge.
- They can be simple to deploy across multiple handsets.
- They offer an improved user experience and a richer design, tapping into device features and offline use.

Cons

The cons of mobile web widgets are:

- They typically require a compatible widget platform to be installed on the device.
- They cannot run in any mobile web browser.
- They require learning additional proprietary, non-web-standard techniques.

Mobile Web Applications

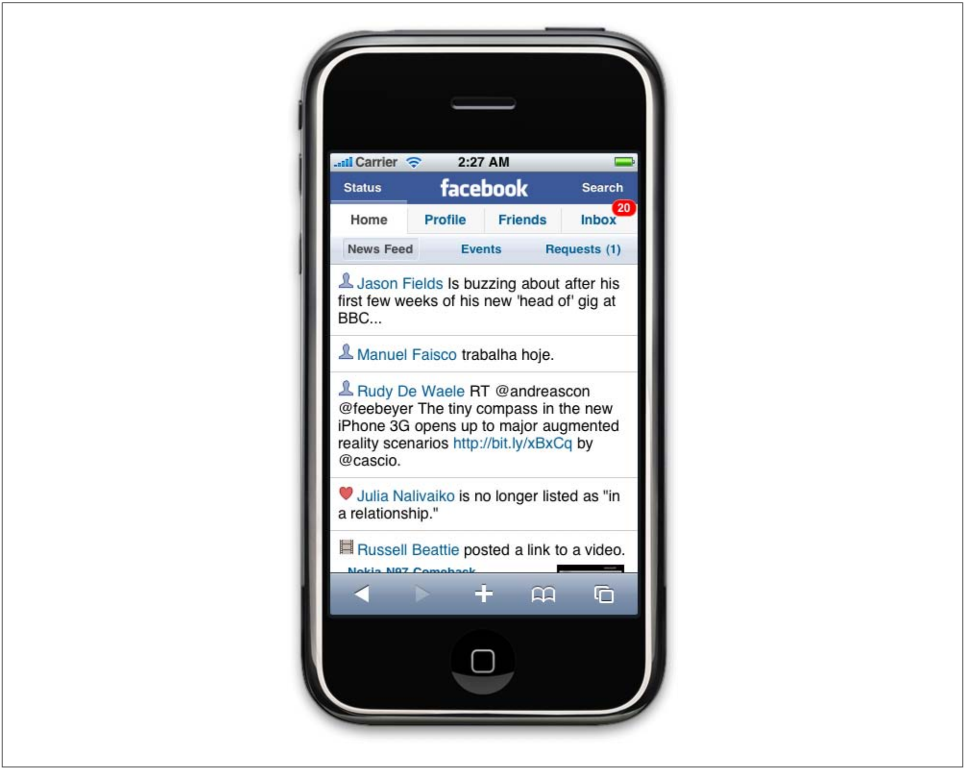
Mobile web applications are mobile applications that do not need to be installed or compiled on the target device. Using XHTML, CSS, and JavaScript, they are able to provide an application-like experience to the end user while running in any mobile web browser. By "application-like" experience, I mean that they do not use the drill-down or page metaphors in which a click equals a refresh of the content in view. Web applications allow users to interact with content in real time, where a click or touch performs an action within the current view.

The history of how mobile web applications came to be so commonplace is interesting, and is one that I think can give us an understanding of how future mobile trends can be assessed and understood. Shortly after the explosion of Web 2.0, web applications like Facebook, Flickr, and Google Reader hit desktop browsers, and there was discussion of how to bring those same web applications to mobile devices. The Web 2.0 movement brought user-centered design principles to the desktop web, and those same principles were sorely needed in the mobile web space as well.

The challenge, as always, was device fragmentation. The mobile browsers were years behind the desktop browsers, making it nearly impossible for a mobile device to render a comparable experience. While XHTML support had become fairly commonplace across devices, the rendering of CSS2 was wildly inconsistent, and support for JavaScript, necessary or simple DHTML, and Ajax was completely nonexistent.

To make matters worse, the perceived market demand for mobile web applications was not seen as a priority with many operators and device makers. It was the classic chicken-or-the-egg scenario. What had to come first, market demand to drive browser innovation or optimized content to drive the market?

With the introduction of the first iPhone, we saw a cataclysmic change across the board. Using WebKit, the iPhone could render web applications not optimized for mobile devices as perfectly usable, including DHTML- and Ajax-powered content. Developers quickly got on board, creating mobile web applications optimized mostly for the iPhone (Figure 6-5). The combination of a high-profile device with an incredibly powerful mobile web browser and a quickly increasing catalog of nicely optimized experiences created the perfect storm the community had been waiting for.



Jason Fields Is buzzing about after his first few weeks of his new 'head of' gig at BBC...

Manuel Faisco trabalha hoje.

Rudy De Waele RT @andreascon @feebeyer The tiny compass in the new iPhone 3G opens up to major augmented reality scenarios http://bit.ly/xBxCq by @cascio.

Julia Nalivaiko is no longer listed as "in a relationship."

Russell Beattie posted a link to a video.

Figure 6-5. The Facebook mobile web app

Usage of the mobile web exploded with not just users of the iPhone, but users of other handsets, too. Because web applications being created for the iPhone were based on web standards, they actually worked reasonably well on other devices. Operators and device makers saw that consumers wanted not just the mobile web on their handsets, but the regular Web, too.

In less than a year, we saw a strong unilateral move by all operators and devices makers to put better mobile web browsers in their phones that could leverage this new application medium. We have not seen such rapid innovation in mobile devices since the inclusion of cameras.

The downside, of course, like all things mobile-web-related, is that not all devices support the capability to render mobile web applications consistently. However, we do see a prevalent trend that the majority of usage of the mobile web is coming from the devices with better browsers, in some markets by a factor of 7:1. So although creating a mobile web application might not reach all devices, it will reach the devices that create the majority of traffic.

Pros

The pros of mobile web applications are:

- They are easy to create, using basic HTML, CSS, and JavaScript knowledge.
- They are simple to deploy across multiple handsets.
- They offer a better user experience and a rich design, tapping into device features and offline use.
- Content is accessible on any mobile web browser.

Cons

The cons of mobile web applications are:

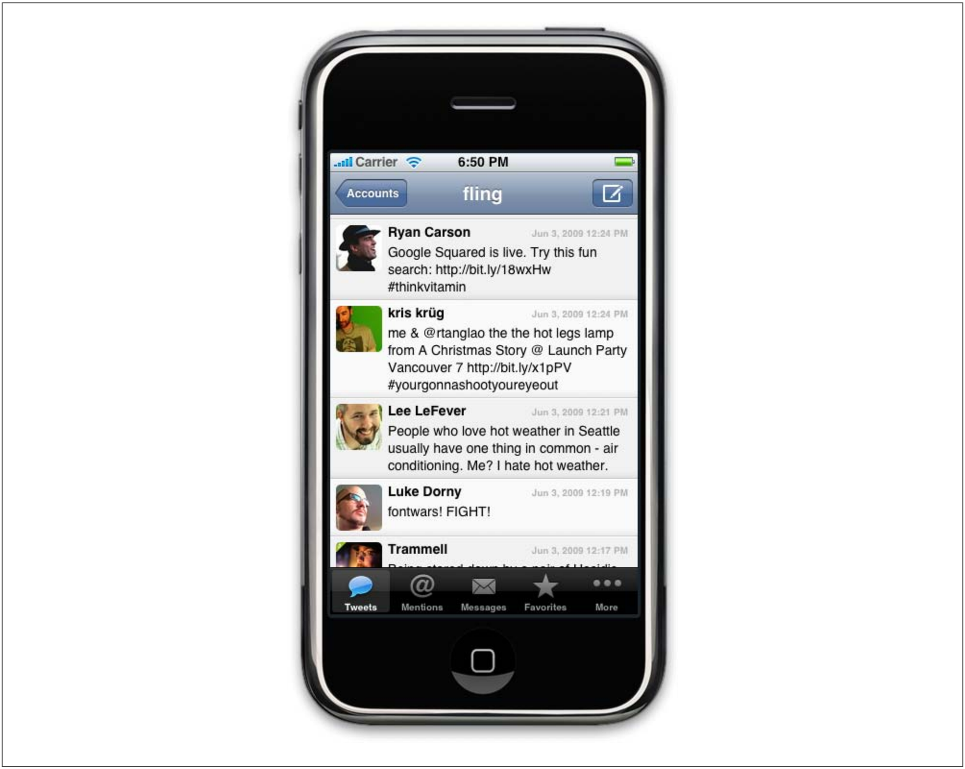
- The optimal experience might not be available on all handsets.
- They can be challenging (but not impossible) to support across multiple devices.
- They don't always support native application features, like offline mode, location lookup, filesystem access, camera, and so on.

Native Applications

The next mobile application medium is the oldest and the most common; it is referred to as native applications, which is actually a misnomer because a mobile web app or mobile web widget can target the native features of the device as well. These applications actually should be called "platform applications," as they have to be developed and compiled for each mobile platform.

These native or platform applications are built specifically for devices that run the platform in question. The most common of all platforms is Java ME (formerly J2ME). In theory, a device written as a Java ME MIDlet should work on the vast majority of feature phones sold around the world. The reality is that even an application written as a Java ME MIDlet still requires some adaptation and testing for each device it is deployed on.

In the smartphone space, the platform SDKs get much more specific. Although many smartphones are also powered by Java, an operating system layer and APIs added to allow developers to more easily offload complex tasks to the API instead of writing methods from scratch. In addition to Java, other smartphone programming languages include versions of C, C++, and Objective-C (Figure 6-6).



Ryan Carson Jun 3, 2009 12:24 PM
Google Squared is live. Try this fun search: http://bit.ly/18wxHw #thinkvitamin

kris krüg Jun 3, 2009 12:24 PM
me & @rtanglao the the hot legs lamp from A Christmas Story @ Launch Party Vancouver 7 http://bit.ly/x1pPV #yourgonnashootyoureyeout

Lee LeFever Jun 3, 2009 12:21 PM
People who love hot weather in Seattle usually have one thing in common - air conditioning. Me? I hate hot weather.

Luke Dorny Jun 3, 2009 12:19 PM
fontwars! FIGHT!

Trammell Jun 3, 2009 12:17 PM

Figure 6-6. A native application in the iPhone

Creating a platform application means deciding which devices to target, having a means of testing and certification, and a method to distribute the application to users. The vast majority of platform applications are certified, sold, and distributed either through an operator portal or an app store. It is possible to create a Java ME MIDlet application and publish it for free on the Web, but it is rarely done.

Because platform applications sit on top of the platform layer, they can tap into the majority of the device features, working online or offline, accessing the location and the filesystem—and if there's camera on the device, then you can probably do something with it as well. Hence the need for certification before the application is distributed, to ensure that no one distributes an application that steals a user's personal data or maliciously uses the device to spread viruses.

However, if you exclude games, the majority (by some estimates, as much as 70 percent) of native applications in use today could be created with a little bit of XHTML, CSS, and JavaScript—in other words, a mobile web application, with little or no feature loss to the user. The advantage is that a mobile application can be developed faster, will work on more devices, require less testing, and be updated more transparently than a native application, which requires third-party certification and publishing in order to get on users' devices. All of these aspects are highly desired in the platform application space. The downside is that it requires a fast and capable mobile web browser that supports offline data and access to device features like location.

Pros

The pros of native applications include:

- They offer a best-in-class user experience, offering a rich design and tapping into device features and offline use.
- They are relatively simple to develop for a single platform.
- You can charge for applications.

Cons

The cons of native applications include:

- They cannot be easily ported to other mobile platforms.
- Developing, testing, and supporting multiple device platforms is incredibly costly.
- They require certification and distribution from a third party that you have no control over.
- They require you to share revenue with the one or more third parties.

Games

The final mobile medium is games, the most popular of all media available to mobile devices. Technically games are really just native applications that use the similar platform SDKs to create immersive experiences (Figure 6-7). But I treat them differently from native applications for two reasons: they cannot be easily duplicated with web technologies, and porting them to multiple mobile platforms is a bit easier than typical platform-based applications.

Although you can do many things with a powerful mobile web browser, creating an immersive gaming experience is not one of them—at least not yet. Seeing as how we have yet to see these types of gaming experiences appear on the desktop using standard web technologies, I believe we are still a few years out from seeing them on mobile devices. Adobe's Flash and the SVG (scalable vector graphics) standard are the only way to do it on the Web now, and will likely be how it is done on mobile devices in the future, the primary obstacle being the performance of the device in dealing with vector graphics.

The reason games are relatively easy to port ("relatively" being the key word), is that the bulk of the gaming experience is in the graphics and actually uses very little of the device APIs. The game mechanics are the only thing that needs to adapted to the various

Figure 6-7. An example game for the iPhone

platforms. Like in console gaming, there are a great number of mobile game porting shops that can quickly take a game written in one language and port it to another.

These differences, in my mind, are what make mobile games stand apart from all other application genres—their capability to be unique and difficult to duplicate in another application type, though the game itself is relatively easy to port. Looking at this model for other application areas—namely, the mobile web—could provide helpful insight into how we create the future of mobile web applications.

Pros

The pros of game applications are:

- They provide a simple and easy way to create an immersive experience.
- They can be ported to multiple devices relatively easily.

Cons

The cons of game applications are:

- They can be costly to develop as an original game title.
- They cannot easily be ported to the mobile web.

Mobile Application Media Matrix

In summary, to aid in comparing and contrasting which of these mobile application media is best for your mobile product, I've placed them into a matrix (Table 6-1).

Table 6-1. Mobile application media matrix

	Device support	Complexity	User experience	Language	Offline support	Device features
SMS	All	Simple	Limited	N/A	No	None
Mobile websites	All	Simple	Limited	HTML	No	None
Mobile web widgets	Some	Medium	Great	HTML	Limited	Limited
Mobile web applications	Some	Medium	Great	HTML, CSS, JavaScript	Limited	Limited
Native applications	All	Complex	Excellent	Various	Yes	Yes
Games	All	Complex	Excellent	Various	Yes	Yes

Application Context

Once your application medium is decided upon, it is time to look at the application context, or the appropriate type of application to present to the user in order for the user to process and understand the information presented and complete her goals. Where the application medium refers mostly to the technical approach of creating an application, the application context deals with the user experience. As discussed in Chapter 4, context is the surroundings in which information is processed, and the application user experience is no different.

Applications can be presented in a variety of ways, ranging from a simple task-based utility to an experience meant to consume the user's focus and attention. There of course is no right or wrong direction—only what is best for your user. In fact, nothing says that you can't use multiple application contexts within the same application—I just wouldn't recommend it unless you have really thought out the flow of your application, because typically it is best to present only one application context so as to avoid confusing the user. If you think it best for your app to mix contexts, then give the user the option to switch between them; for example, some smartphones allow for an orientation change, so if the device is rotated to landscape mode, your app switches from an informative view to a utility view, or maybe from a locale view to an immersive view.

Utility Context

The most basic application context is the utility, or a simple user experience metaphor that is meant to address short, task-based scenarios. Information is meant to be presented in a minimal fashion, often using the least amount of user input as possible. An example of a utility might be a calculator, weather forecast, unit conversion, stocks, world clock, and so on. In each of these cases, the user enters a small amount of information into the utility, like a simple equation, a city, or a stock symbol, and either by performing a small background task or fetching information online, the utility is able to present data to the user in the desired context (Figure 6-8).

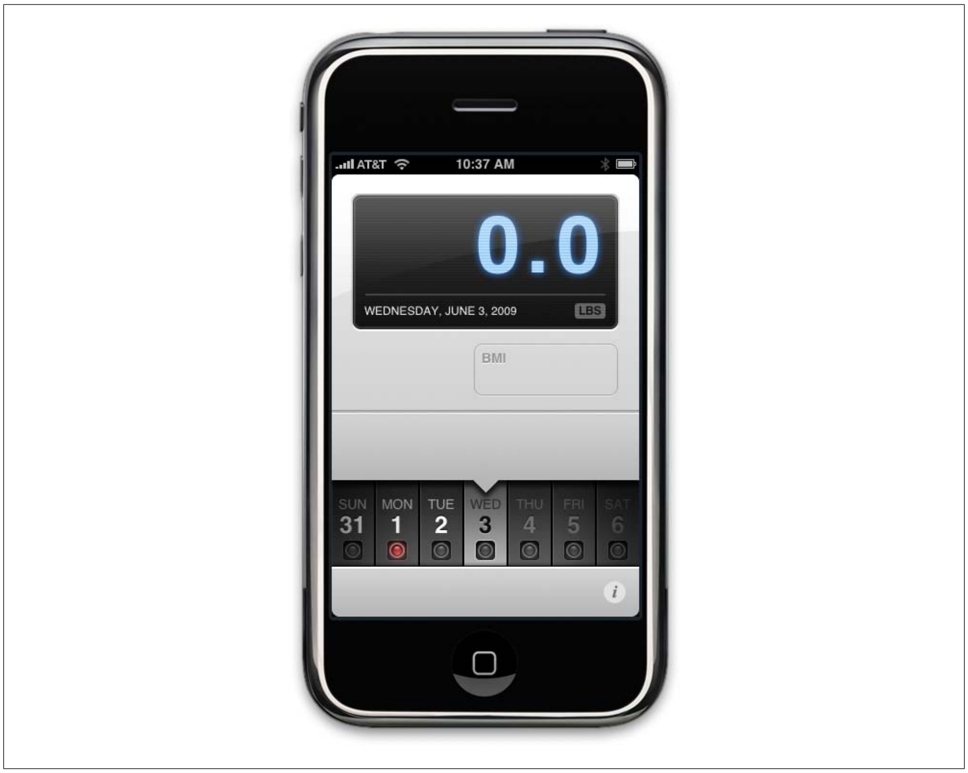


Figure 6-8. An example utility application

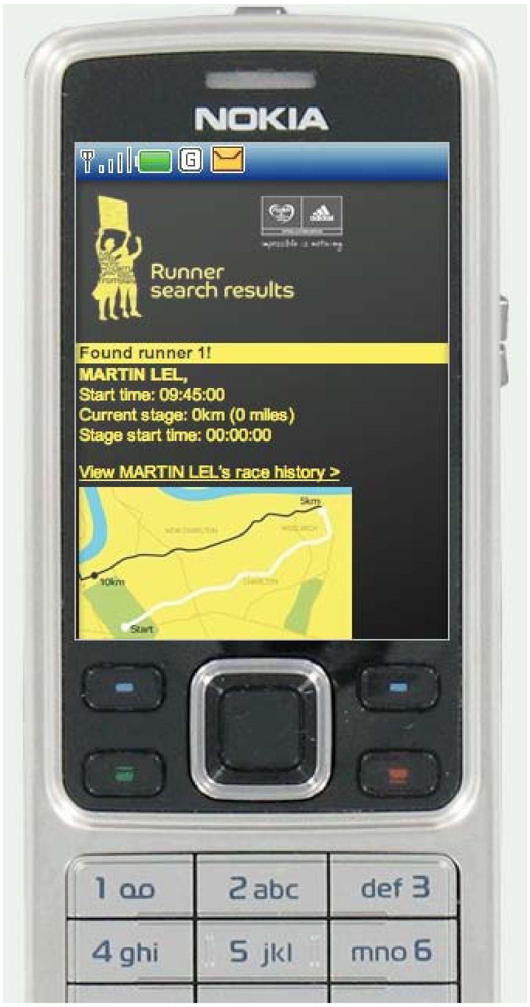
The goal of the utility is to give users at-a-glance information, therefore offering users a minimal design aesthetic, focusing the design around the content in view, and often using larger type and a sparse layout.

It would be easy to mistake utilities for widgets, given that widgets are a "component of a user interface that operates in a particular way." But utilities can be much more than widgets; they are not merely an extension of the user experience, but are applications in their own right that can establish their own look and feel separate from the established user experience.

Use utilities for short, simple tasks, at-a-glance information, when there is limited content to display, and when combined with an immersive context to create dual-mode applications.

Locale Context

The locale context is a newer application type that is still being defined by the mobile community, but we are certainly seeing some clear patterns of how to create locale applications (Figure 6-9). As more location information is being published online, and more devices add GPS to pinpoint the user's location, locale is becoming an excellent data point to pivot information around. For example, I can use location to display the cafés nearest to my current location. Plus I can layer multiple data sources into the application, such as: of the cafés nearest to me, which ones have free wireless access? Or, do I have friends in the general area that can meet me?



Found runner 1!
MARTIN LEL,
Start time: 09:45:00
Current stage: 0km (0 miles)
Stage start time: 00:00:00

View MARTIN LEL's race history >

Figure 6-9. An example locale application

Locale applications almost always have at least one thing in common: a map on which they plot the requested data points visually. At the very least, they list items in order of distance, with the nearest item first and the farthest last. The user's goal is to find information relative to his present location, and content should always be designed with this in mind. When creating locale apps, it is important to ensure that the user's present location is always clearly identified, as well as a means of adding data to it. This could be another location, in the case of finding point-to-point directions, or it could be a keyword query to find people, places, or things nearby.

Use locale applications for location-based applications, applications that might contain a dynamic map, and listing multiple location-based points of interest.

Informative Applications

The informative application is an application context in which the one and only goal is to provide information, like a news site, an online directory, a marketing site, or even a mobile commerce site, where the key task of the user is to read and understand and it is not necessary to interact (Figure 6-10). This isn't to say that you cannot include calls to action in the informative context—in fact, you should, but they should be based around what you can assume about your users in this context.

For example, remember that most mobile tasks are short and are often undertaken during brief idle periods. The user doesn't have much extra time and the task can be interrupted at any moment. In the case of a mobile news site, provide the user with the option to mark a page or story to be read later. With an online directory, allow the user to flag favorite entries. With a marketing site, allow users to enter the shortest possible contact information, like their phone number or email. And with a mobile commerce site, allow users to save items to a wishlist to review and purchase later.

The theme here is that although reading information is a simple task, it usually creates a complex chain of events that can be anticipated. With mobile applications, we want to avoid forcing the users to input too much information with their mobile devices, which is more difficult and takes more time than it would on another medium such as a desktop or laptop computers. Instead, we want to look for ways we can interconnect experiences, having users use the informative context to filter to the most desirable information when they have a moment, and allowing them to interact with it later, when they have more time, from the medium of their choice.

Use informative applications when users need information to gain an understanding or perform a physical task, for applications that can be used in multimedia contexts such as desktop and mobile, for information-heavy applications, and for marketing or promotional applications.

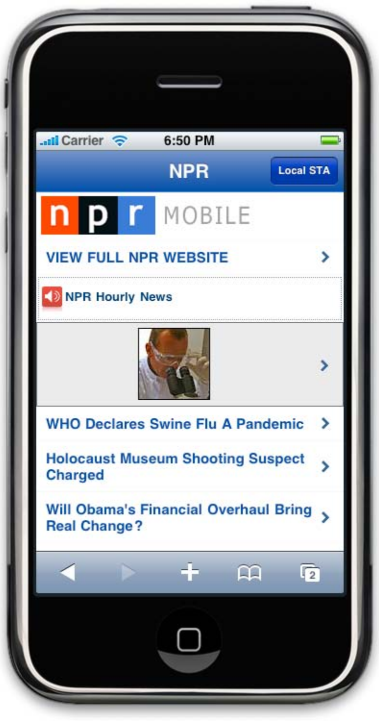
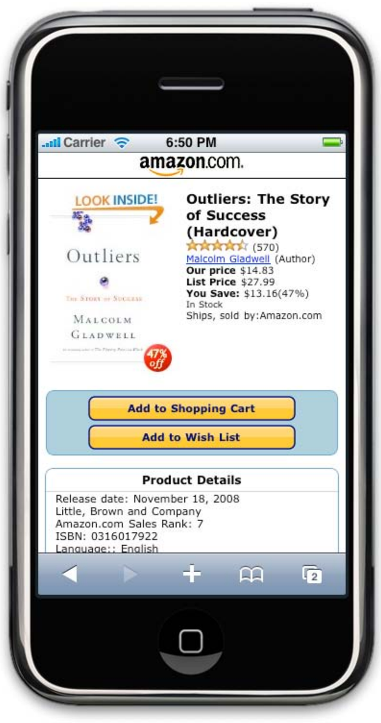

Carrier
6:50 PM
NPR
Local STA
n p r MOBILE
VIEW FULL NPR WEBSITE
NPR Hourly News
WHO Declares Swine Flu A Pandemic
Holocaust Museum Shooting Suspect Charged
Will Obama's Financial Overhaul Bring Real Change?

Figure 6-10. An example informative application

Productivity Application Context

The productivity application context is used for content and services that are heavily task-based and meant to increase the users' sense of efficiency. With these types of applications, we can assume that the users are more committed to accomplishing a particular goal, like managing content such as messages, contacts, or media, but we should still assume that they are doing so during idle periods (Figure 6-11). Just because the application context is meant to make users more productive, we can't assume that they are able to make the same time commitment as they would in the desktop context.

Productivity applications are often very structured, presenting information in a defined hierarchy and often using the folder or group metaphor to define a sense of order to the user. When designing these types of apps, we need to pay careful consideration to how the user thinks out the task. People have an uncanny ability to understand and recall complex hierarchies of tasks—for example, what they need to do first, second, and third in order for a particular solution to work. We take this for granted and in the desktop context often show the users the entire hierarchy visually. In the mobile context, we don't have the screen real estate, and therefore need to help users find their way.



Outliers: The Story of Success (Hardcover)
(570)
Malcolm Gladwell (Author)
Our price $14.83
List Price $27.99
You Save: $13.16(47%)
In Stock
Ships, sold by:Amazon.com

Product Details
Release date: November 18, 2008
Little, Brown and Company
Amazon.com Sales Rank: 7
ISBN: 0316017922
Language:: English

Figure 6-11. An example productivity application

One method is to focus on prioritization of tasks; productivity applications typically include some method of direct or indirect prioritization. If we look at a mobile email client, we see that the app generally focuses around the inbox, which is the top-priority item, given that all new messages will come there first. All other folders are of a lower priority, as in order for messages to get there, we will have had to process them previously. We can use this screen as a central focus point, assuming that users will spend the majority of their time there, and branching out onto other screens from this central spot. But we can't forget other high-priority items, like the ability to send a new message or create a new item. This is a task that is typically included on every screen within an email app, and in the same position throughout to ensure that users always have quick access to create a new item.

The productivity context is one of the hardest application contexts to get right, so do yourself a favor and start simple. Start with one feature, treat it almost as if it were a single focus utility, and get it right before you move on to the next. Layer in your features one at a time until you feel like you have met the users' goals, and stop the moment that it becomes it bit overwhelming to manage. You probably won't be able to include every feature, so you will need to include only the ones that are most important to users, and lose everything else.

Use the productivity application context for information-heavy applications where the user will need to manage content from a mobile device and for heavily structured, hierarchy-based tasks.

Immersive Full-Screen Applications

The final application context is an immersive full-screen application, like a game, a media player, or possibly even a single-screen utility. These applications are meant to consume the user's focus, often doing so by filling the entire screen (Figure 6-12), and leaving no trace of the device user interface to distract the user. Again, the majority of mobile engagement occurs when the user has idle periods of time; the immersive context is typical in most entertainment applications, one of the most popular mobile content areas.

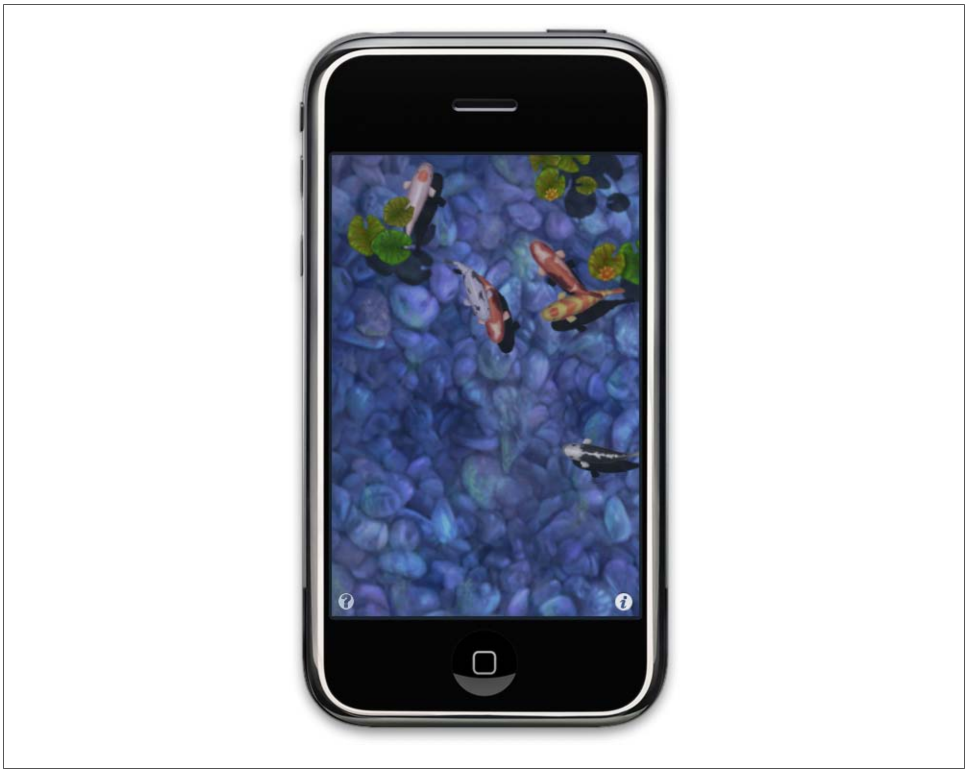


Figure 6-12. An example of an immersive application

The most common use of the immersive context is obviously with a game, for which you want the user to focus on how to play the game. But this context can also be used with other contexts, presenting a full-screen view of content when the device orientation changes in many higher-end devices. For example, if we were making a locale-based application, we could add a feature that changes the user experience to the immersive context, showing a full-screen map, or point-by-point directions, whenever the device is held in landscape view. This is not a feature that many applications include, but I think it is worth considering.

Even with mobile web apps, many devices allow for detection of an orientation change. Typically, the app just scales to fill the page, actually breaking the intended user experience, but by adding the orientation-specific styles, the designer could create an immersive version of the application, presenting the app content in a more at-a-glance, friendly way, helpful for devices placed on automobile dashboards, or held in the hand to show others.

Use an immersive full-screen application for games, media players, and alternative views of another application context.

Application Context Matrix

I put each of the application contexts into Table 6-2, comparing and contrasting their benefits to help you determine what is best for your application.

Table 6-2. Application context matrix

	User experience type	Task type	Task duration	Combine with
Utility	At-a-glance	Information recall	Very short	Immersive
Locale	Location-based	Contextual information	Quick	Immersive
Informative	Content-based	Seek information	Quick	Locale
Productivity	Task-based	Content management	Long	Utility
Immersive	Full screen	Entertainment	Long	Utility, locale

As you can see, mobile applications can run the gamut from intense experiences to simple tools. In some cases, they can switch back and forth between the two. Figure out which type of application is best for your users and in what context.

CHAPTER 7

Mobile Information Architecture

Your information architecture (also known as IA), is the foundation of your mobile product. A well-engineered product with good visual design can still fail because of poor information architecture. The truly successful mobile products always have a well-thought-out information architecture.

From a simple mobile website to an iPhone application, the mobile information architecture defines not just how your information will be structured, but also how people will interact with it. This is made especially tricky when you consider that different devices have different capabilities and therefore different interaction models. Take the way people interact with their devices: for example, a touch device on which the user literally points and clicks, or a more basic device on which the user uses the directional pad to navigate to the desired location.

Also, don't forget that the information architecture emphasizes how you address context. In other words, a good mobile information architecture is based around the various user contexts that I talked about in Chapter 4. The secret is that mobile information architecture isn't all that different from how you might architect software or a website; it just has a few added challenges.

What Is Information Architecture?

Before we get into the specifics of mobile information architecture, let's first talk about exactly what information is. I can think of no better definition than the seminal O'Reilly book *Information Architecture for the World Wide Web* (*http://oreilly.com/catalog/9780596527341/*) by Morville and Rosenfeld, otherwise known in information architect circles as the "polar bear" book. This definition outlines the following:

- The structural design of shared information environments
- The combination of organizations, labeling, search, and navigation systems within websites and intranets

- The art and science of shaping information products and experiences to support usability and findability
- An emerging discipline and community of practice focused on bringing principles of design and architecture to the digital landscape

Similar to how mobile technology has many facets, so does information architecture, as it is often used as an umbrella term to describe several unique disciplines, including the following:

Information architecture
: The organization of data within an informational space. In other words, how the user will get to information or perform tasks within a website or application.

Interaction design
: The design of how the user can participate with the information present, either in a direct or indirect way, meaning how the user will interact with the website of application to create a more meaningful experience and accomplish her goals.

Information design
: The visual layout of information or how the user will assess meaning and direction given the information presented to him.

Navigation design
: The words used to describe information spaces; the labels or triggers used to tell the users what something is and to establish the expectation of what they will find.

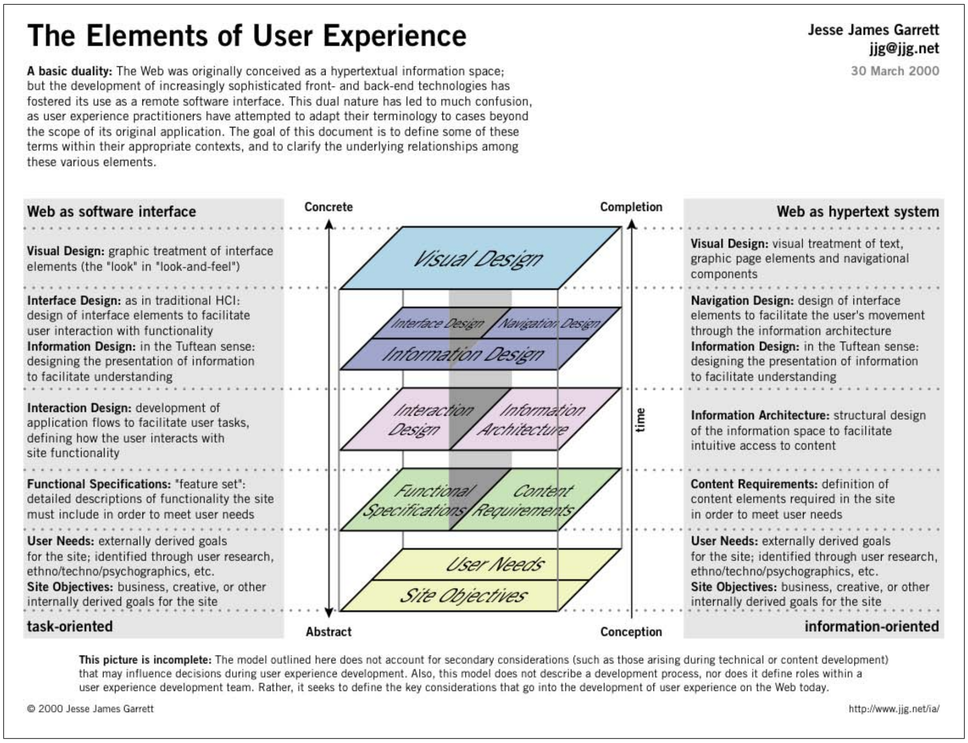
Interface design
: The design of the visual paradigms used to create action or understanding.

In 2000, information architect Jesse James Garrett created the Elements of User Experience (*http://www.jjg.net/elements/pdf/elements.pdf*, also seen in Figure 7-1),[*] to help show how each of the layers of the user experience come together to create both hypertext experiences and the more software-like application experience.

As you can see, information architecture composes the core of the user experience. The role of information architecture is played by a variety of people, from product managers to designers and even developers. To make things more confusing, information architecture can be called many different things throughout the design and development process. Words like intuitive, simple, findable, usable, or the executive favorite—easy-to-use—all describe the role that information architects play in creating digital experiences.

The visual design of your product, what frameworks you use, and how it is developed are integral to the success of any product, but the information architecture stands apart as being the most crucial element of your product. It is the first line of scrimmage—the user's first impression of your product. Even if you have the best design, the best code,

[*] Image used with permission of Jesse James Garrett (*http://jjg.net/elements/*).



The Elements of User Experience

Jesse James Garrett
jjg@jjg.net
30 March 2000

A basic duality: The Web was originally conceived as a hypertextual information space; but the development of increasingly sophisticated front- and back-end technologies has fostered its use as a remote software interface. This dual nature has led to much confusion, as user experience practitioners have attempted to adapt their terminology to cases beyond the scope of its original application. The goal of this document is to define some of these terms within their appropriate contexts, and to clarify the underlying relationships among these various elements.

Web as software interface

Visual Design: graphic treatment of interface elements (the "look" in "look-and-feel")

Interface Design: as in traditional HCI: design of interface elements to facilitate user interaction with functionality
Information Design: in the Tuftean sense: designing the presentation of information to facilitate understanding

Interaction Design: development of application flows to facilitate user tasks, defining how the user interacts with site functionality

Functional Specifications: "feature set": detailed descriptions of functionality the site must include in order to meet user needs

User Needs: externally derived goals for the site; identified through user research, ethno/techno/psychographics, etc.
Site Objectives: business, creative, or other internally derived goals for the site

task-oriented

Web as hypertext system

Visual Design: visual treatment of text, graphic page elements and navigational components

Navigation Design: design of interface elements to facilitate the user's movement through the information architecture
Information Design: in the Tuftean sense: designing the presentation of information to facilitate understanding

Information Architecture: structural design of the information space to facilitate intuitive access to content

Content Requirements: definition of content elements required in the site in order to meet user needs

User Needs: externally derived goals for the site; identified through user research, ethno/techno/psychographics, etc.
Site Objectives: business, creative, or other internally derived goals for the site

information-oriented

This picture is incomplete: The model outlined here does not account for secondary considerations (such as those arising during technical or content development) that may influence decisions during user experience development. Also, this model does not describe a development process, nor does it define roles within a user experience development team. Rather, it seeks to define the key considerations that go into the development of user experience on the Web today.

© 2000 Jesse James Garrett

http://www.jjg.net/ia/

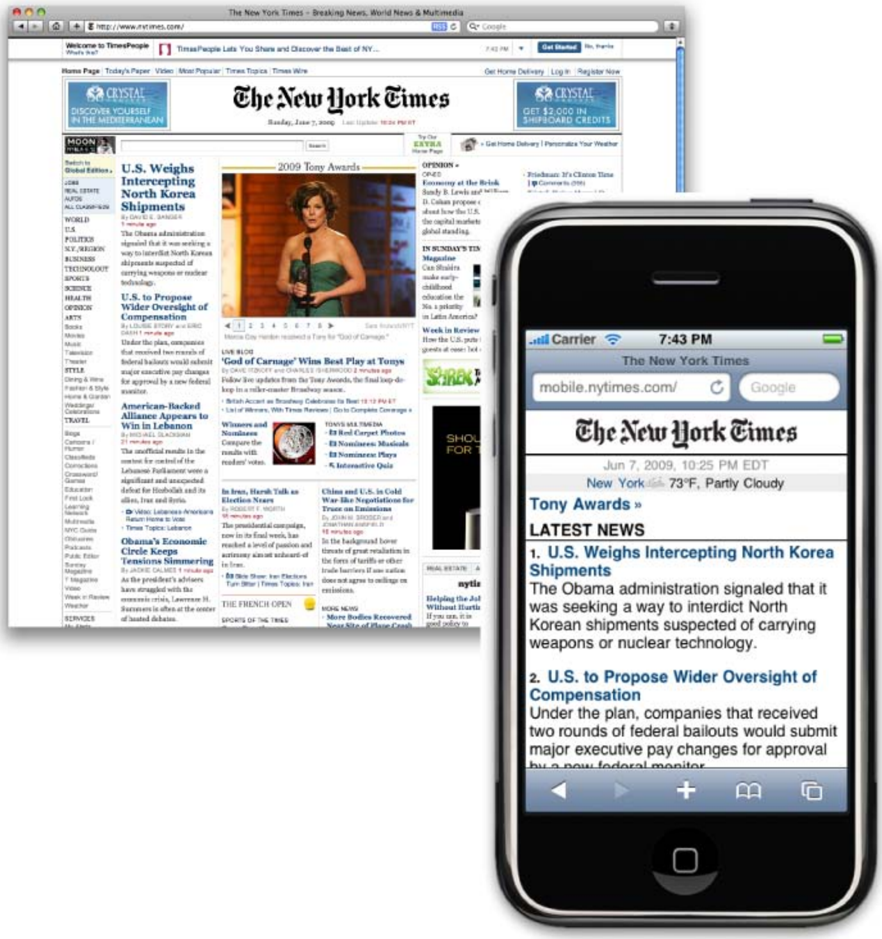
Figure 7-1. Jesse James Garrett's Elements of User Experience

and the best backend service, if the user cannot figure out how to use it, she will fail—and so will your product.

Mobile Information Architecture

Although information architecture has become a common discipline in the web industry, unfortunately, the mobile industry—like software—has only a handful of specialized mobile information architects. Although mobile information architecture is hardly a discipline in its own right, it certainly ought to be. This is not because it is so dissimilar from its desktop cousin, but because of context, added technical constraints, and needing to display on a smaller screen as much information as we would on a desktop.

For example, if we look at the front page of *http://www.nytimes.com* as seen from a desktop web browser compared to how it may render in a mobile browser (Figure 7-2), we see a content-heavy site that works well on the desktop, and is designed to present the maximum amount of information above the "fold" or where the screen cuts off the content. However, in the mobile browser, the text is far too small to be useful.


The New York Times - Breaking News, World News & Multimedia

The New York Times

U.S. Weighs Intercepting North Korea Shipments

U.S. to Propose Wider Oversight of Compensation

American-Backed Alliance Appears to Win in Lebanon

Obama's Economic Circle Keeps Tensions Simmering

2009 Tony Awards

'God of Carnage' Wins Best Play at Tonys

Carrier 7:43 PM

The New York Times

mobile.nytimes.com/

Google

The New York Times

Jun 7, 2009, 10:25 PM EDT

New York 73°F, Partly Cloudy

Tony Awards »

LATEST NEWS

1. U.S. Weighs Intercepting North Korea Shipments

The Obama administration signaled that it was seeking a way to interdict North Korean shipments suspected of carrying weapons or nuclear technology.

2. U.S. to Propose Wider Oversight of Compensation

Under the plan, companies that received two rounds of federal bailouts would submit major executive pay changes for approval

Figure 7-2. *Comparing the New York Times website in desktop and mobile browsers*

The role of a mobile information architect would be to interpret this content to the mobile context. Do you use the same structure, or sections? Do you present the same information above the fold? If so, how should that be prioritized? How does the user navigate to other areas? Do you use the same visual and interaction paradigms, or invent new ones? And if you do start to invent new paradigms, will you lose the visual characteristics of what users expect?

These are only some of the questions you have to ask yourself when starting to create a mobile information architecture. As you can see in Figure 7-3 there are several different ways that the *New York Times* has been interpreted for the mobile context.

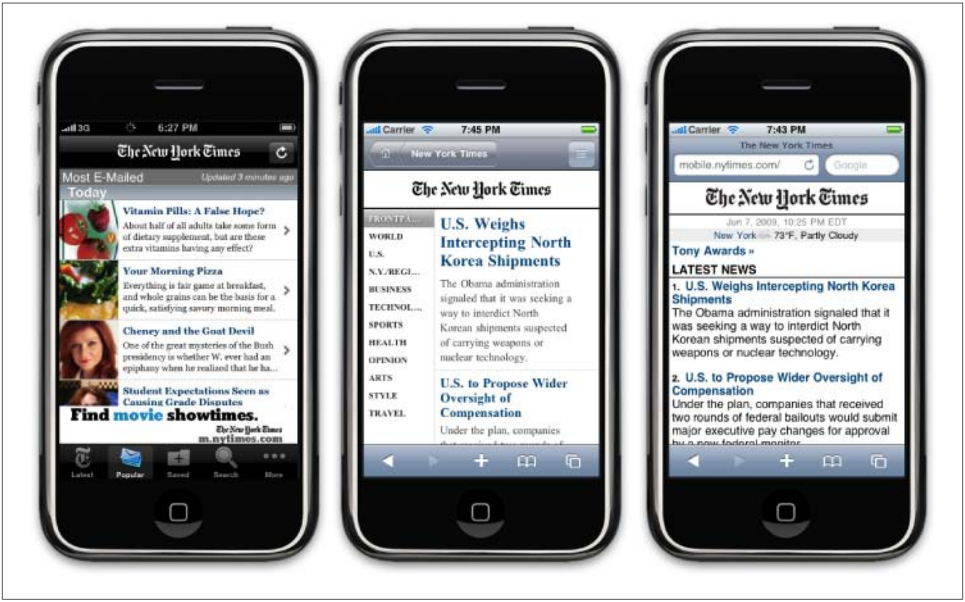


Figure 7-3. *The many mobile experiences of the New York Times*

But we are only beginning to scratch the surface. We also need to design our mobile information architecture to address the mobile context. Given that many devices can detect our current location, which is one of the most immediate types of context, how does the *New York Times* application address the user's context? For example, as a publication that serves both New York City and a larger global audience, if I'm not in New York, should I still see the local New York headlines? Or should I see the headlines based on my location?

Keeping It Simple

When thinking about your mobile information architecture, you want to keep it as simple as possible.

Support your defined goals

If something doesn't support the defined goals, lose it. Go back to your user goals and needs, and identify the tasks that map to them. Find those needs and fill them.

Ask yourself: what need does my application fill? What are people trying to do here? What is their primary goal? Once you understand that, it is a simple process of reverse-engineering the path from where they want to be to where they are starting. Cut out everything else—your site or application doesn't need it. For example, to get some news and information on a mobile device, you need to first ask what the goal is. What is the need you are trying to fill? Then you need to apply context. Where are your users? What are they doing? Are they waiting for the bus? Do they have only a minute to spare? Or, do they have five minutes to spare? With these answers, you get your information architecture.

Clear, simple labels

Good trigger labels, the words we use to describe each link or action, are crucial in Mobile. Words like "products" or "services" aren't good trigger labels. They don't tell us anything about that content or what we can expect. Now, I would argue that good trigger labels are crucial in the Web as well, that we've become lazy and we assume so much about the user that we ignore the use of good trigger labels. Users have a much higher threshold of pain when clicking about on a desktop site or application, hunting and pecking for tasty morsels. Mobile performs short, to-the-point, get-it-quick, and get-out types of tasks. What is convenient on the desktop might be a deal breaker on mobile.

Keep all your labels short and descriptive, and never try to be clever with the words you use to evoke action. The worst sin is to introduce branding or marketing into your information architecture; this will just serve to confuse and distract your users. Executives love to use the words they use internally to external communications on websites and applications, but these words have no meaning outside of your company walls. If the user is just trying to get music, don't call it "My Music," "My MP3s," or something made up that only strokes our corporate egos, such as "AudioJams™"—just call it "Music."

Don't try to differentiate your product offering by what you call it. Create something unique by creating a usable and intuitive experience based on focusing on what users need and using the same language they use to describe those needs.

Based on what we know from web design, you should use simple, direct terms for navigating around your pages rather than overly clever terms. That latter typically result in confused visitors who struggle to find the content they are looking for. When that happens, they will go elsewhere to look for the information they want. So, if you apply these same mistakes to a constrained device like mobile, then you end up adding confusion to the user experience at a higher magnitude than the Web.

Site Maps

The first deliverable we use to define mobile information architecture is the site map. Site maps are a classic information architecture deliverable. They visually represent the relationship of content to other content and provide a map for how the user will travel through the informational space, as shown in Figure 7-4.

Mobile site maps aren't that dissimilar from site maps we might use on the Web. But there are a few tips specific to mobile that we want to consider.

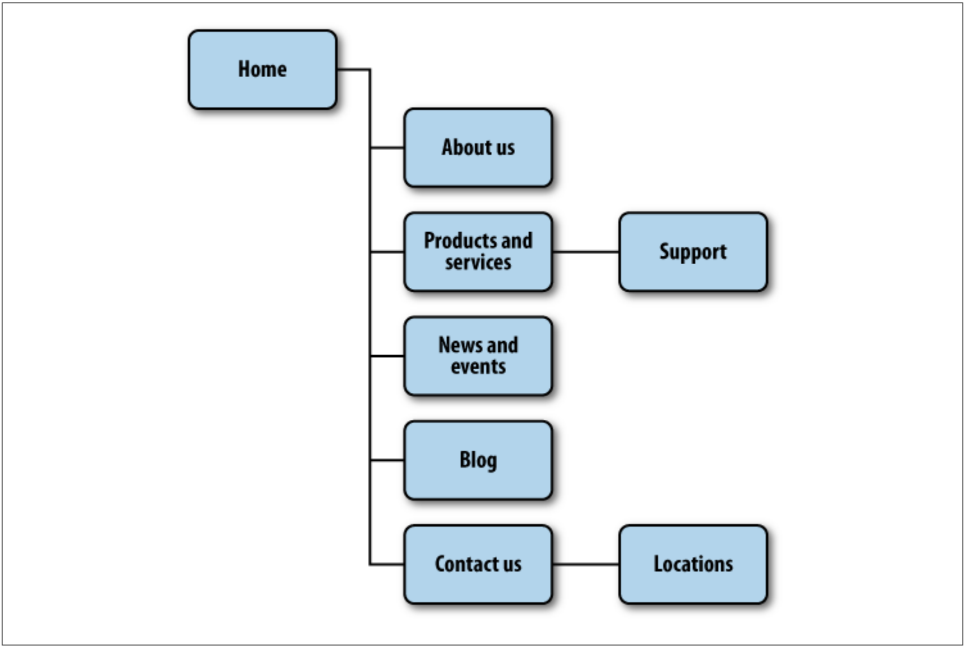


Figure 7-4. An example mobile site map

Limit opportunities for mistakes

Imagine a road with a fork in it. We can go either left or right. The risk that we will make the wrong choice is only 50 percent, meaning that we have a better than good chance that we will get to where we want to go. But imagine three roads. Now our chances have dropped to 33 percent. Four roads drops your chances to 25 percent, and five roads takes you down to 20 percent. Now a 20 percent chance isn't great, but it isn't too bad, either.

Now think of your own website. How many primary navigation areas do you have? Seven? Eight? Ten? Fifteen? What risk is there to the users for making a wrong choice? If they go down the wrong path, they can immediately click back to where they started and go down another path, eliminating the wrong choices to find the right ones. The risks for making the wrong choice are minor.

In Figure 7-5, you can see a poorly designed mobile information architecture that too closely mimics its desktop cousin; it was not designed with the mobile user in mind.

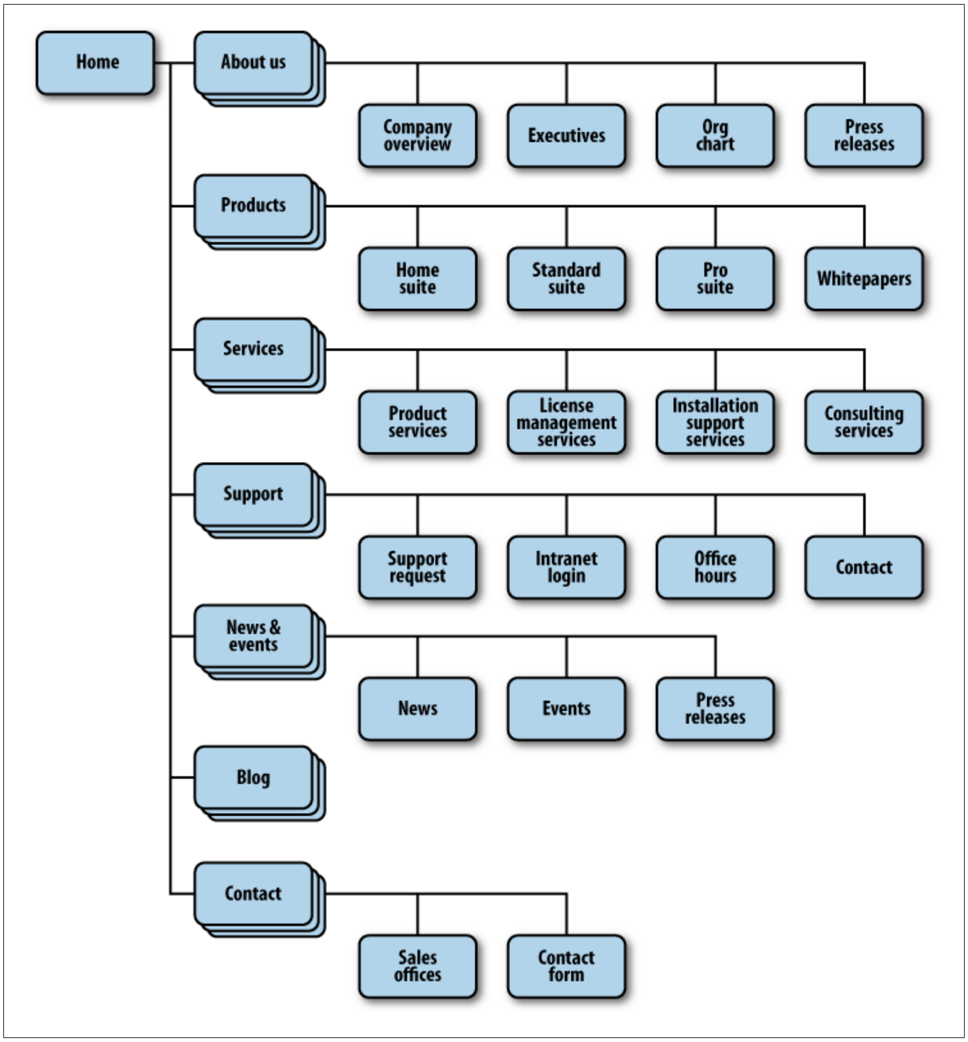


Figure 7-5. An example of a bad mobile information architecture that was designed with desktop users in mind rather than mobile users

But in mobile, we cannot make this assumption. In the mobile context, tasks are short and users have limited time to perform them. And with mobile websites, we can't assume that the users have access to a reliable broadband connection that allows them to quickly go back to the previous page. In addition, the users more often than not have to pay for each page view in data charges. So not only do they pay cash for viewing the wrong page by mistake, they pay to again download the page they started from: we can't assume that pages will be cached properly.

Therefore, my advice is to limit users' options, those forks in the road, to five or less. Anything more, and you introduce far too much risk that the user will make a mistake and head off in the wrong direction.

Confirm the path by teasing content

After the users have selected a path, it isn't always clear whether they are getting to where they need to be. Information-heavy sites and applications often employ nested or drill-down architectures, forcing the user to select category after category to get to their target. To reduce risking the user's time and money, we want to make sure we present enough information for the user to wade through our information architecture successfully. On the Web, we take these risks very lightly, but with mobile, we must give our users a helping hand. We do this by teasing content within each category—that is, providing at least one content item per category.

I've done a fair share of mobile sites and applications that sell ringtones, the historical mobile content motherlode. The challenge with ringtone sites is you have a lot of items, grouped by artist, album, genre, and so on. The user starts with a goal like "I want a new ringtone" and finds an item that suits his taste within a catalog of tens of thousands of items.

In order to make sense of a vast inventory of content, we have to group, subgroup, and sometimes even subgroup again, creating a drill-down path for the user to browse. Though on paper this might seem like a decent solution, once you populate an application with content, the dreaded "Page 1 of 157" appears. What user would ever sit there with a mobile device and page through 157 pages of ringtones? What user would page through five pages of content?

On an early site I worked on, users would flip through a few pages of content, then give up or go back and visit another area. We could see a direct relationship to the number of pages viewed to sales—essentially, more pages loaded meant fewer sales. Then we realized we could take the most popular item based on sales and place it as the first item in any list, which is teasing the content.

In Figure 7-6, you can see in a constrained screen that teasing the first few items of the page provides the user with a much more intuitive interface, immediately indicating what type of content the user can expect.

We immediately saw that users were finding content more quickly, driving up our sales. It was like night and day. Since those days, I've tested this principle on a variety of mobile sites—not just ringtones, but game catalogs, news sites, and regular old corporate websites. Each time, it has improved the conversion, getting users to the content they seek with the least amount of backtracking.

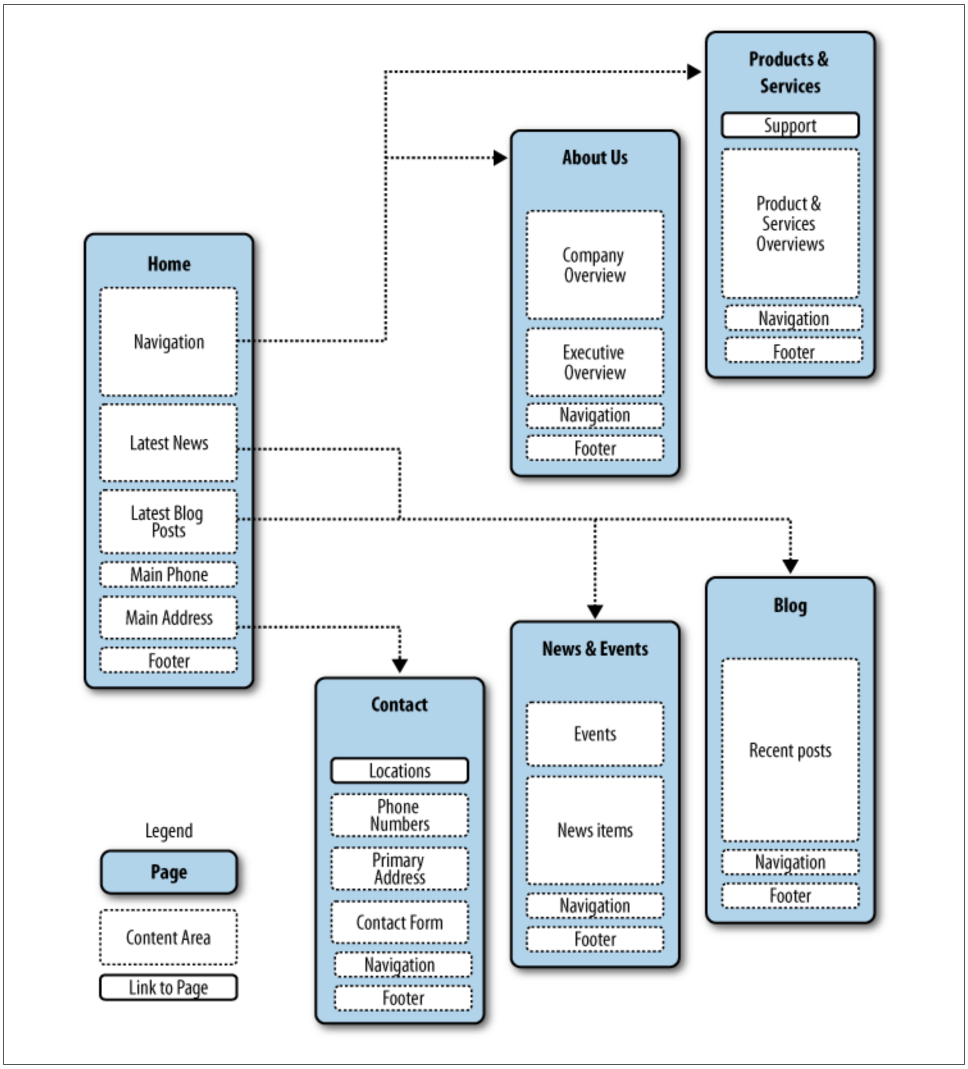


Figure 7-6. Teasing content to confirm the user's expectations of the content within

Clickstreams

Clickstream is a term used for showing the behavior on websites, displaying the order in which users travel through a site's information architecture, usually based on data gathered from server logs. Clickstreams are usually historical, used to see the flaws in your information architecture, typically using heat-mapping or simple percentages to show where your users are going. I've always found them to be a useful tool for rearchitecting large websites.

However, information architecture in mobile is more like software than it is the Web, meaning that you create clickstreams in the beginning, not the end. This maps the ideal path the user will take to perform common tasks. Being able to visually lay out the path users will take gives you a holistic or bird's-eye view of your mobile information architecture, just as a road map does. When you can see all the paths next to each other and take a step back, you start to see shortcuts and how you can get users to their goal faster or easier, as shown in Figure 7-7.

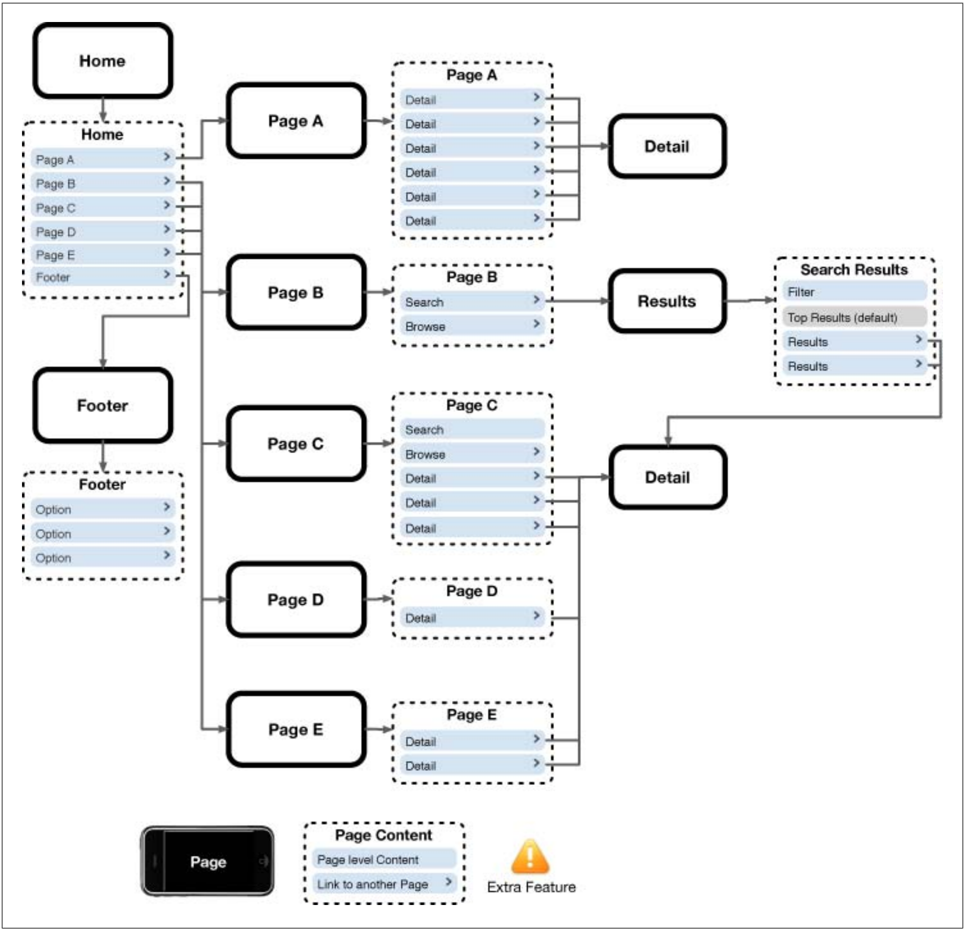


Figure 7-7. An example clickstream for an iPhone web application

Now the business analyst in you is probably saying, "Just create user or process flows," such as the esoteric diagram shown in Figure 7-8, which is made up of boxes and diamonds that look more like circuit board diagrams than an information architecture.

If that is what your team prefers, then by all means, flow away. Personally, I like to present all of my information architecture deliverables from the perspective of the user,

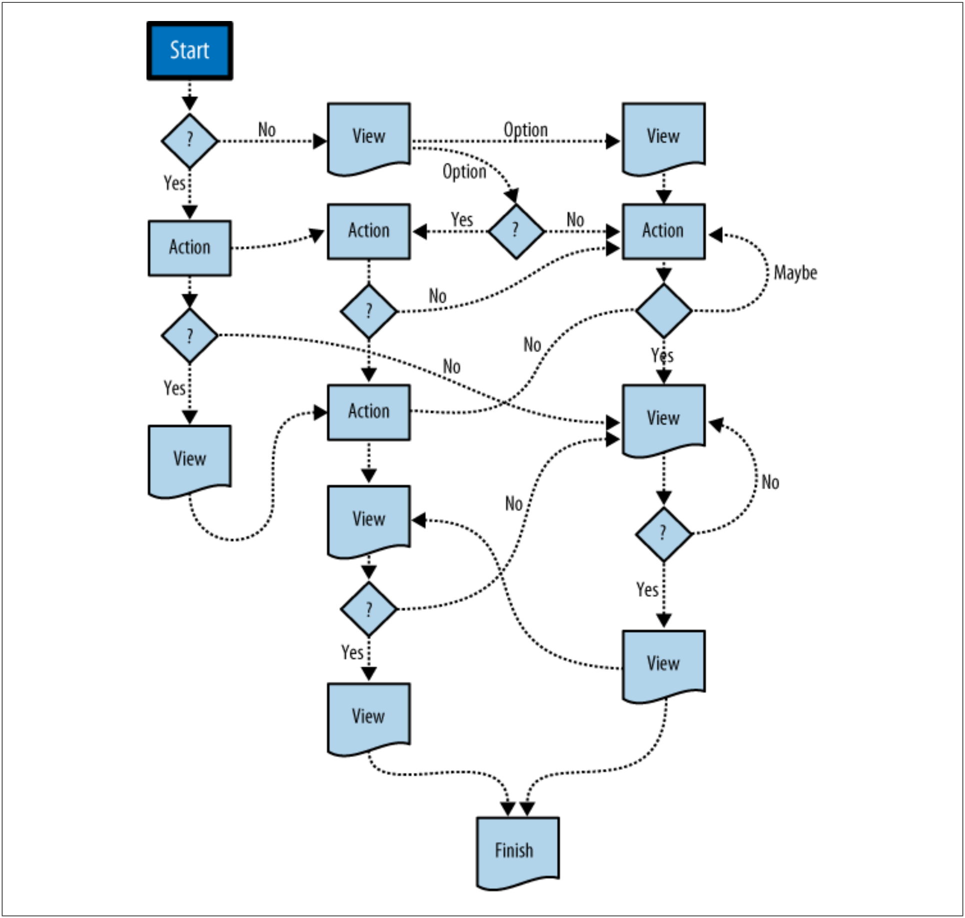


Figure 7-8. An example process flow diagram

using the same metaphors she will use to make her way through my information architecture—in this case, either a screen or page metaphor.

A good architect's job is to create a map of user goals, not map out every technical contingency or edge case. Too often, process flows go down a slippery slope of adding every project requirement, bogging down the user experience with unnecessary distractions, rather than focusing on streamlining the experience. Remember, in mobile, our job is to keep it as simple as possible. We need to have an unwavering focus on defining an excellent user experience first and foremost. Anything that distracts us from that goal is just a distraction.

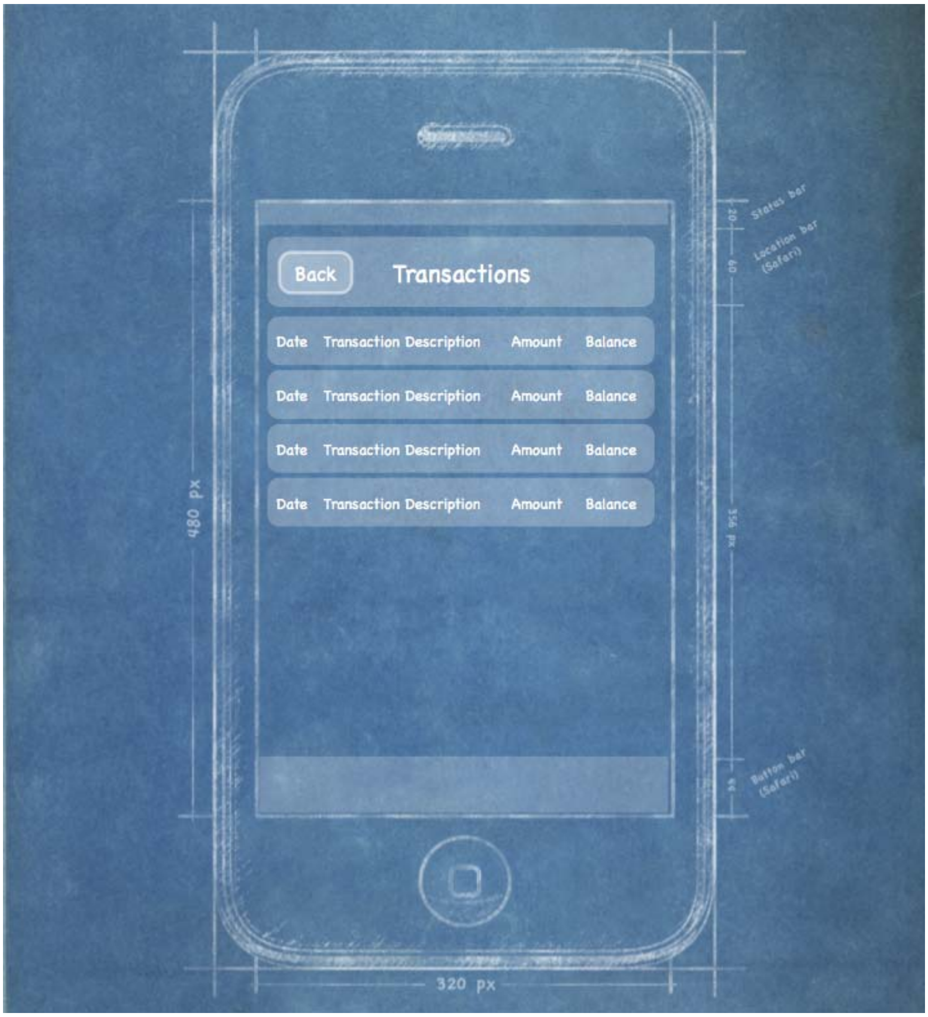

Back
Transactions
Date Transaction Description Amount Balance
Date Transaction Description Amount Balance
Date Transaction Description Amount Balance
Date Transaction Description Amount Balance
480 px
320 px
356 px
Status bar
Location bar (Safari)
Button bar (Safari)

Figure 7-9. An example of an iPhone web application wireframe, intended to be low fidelity to prevent confusion of visual design concepts with information design concepts

Wireframes

The next information architecture tool at our disposal is wireframes. *Wireframes* are a way to lay out information on the page, also referred to as *information design*. Site maps show how our content is organized in our informational space; wireframes show how the user will directly interact with it. Wireframes are like the peanut butter to the site map jelly in our information architecture sandwich. It's the stuff that sticks. Wireframes like the one in Figure 7-9 serve to make our information space tangible and useful.

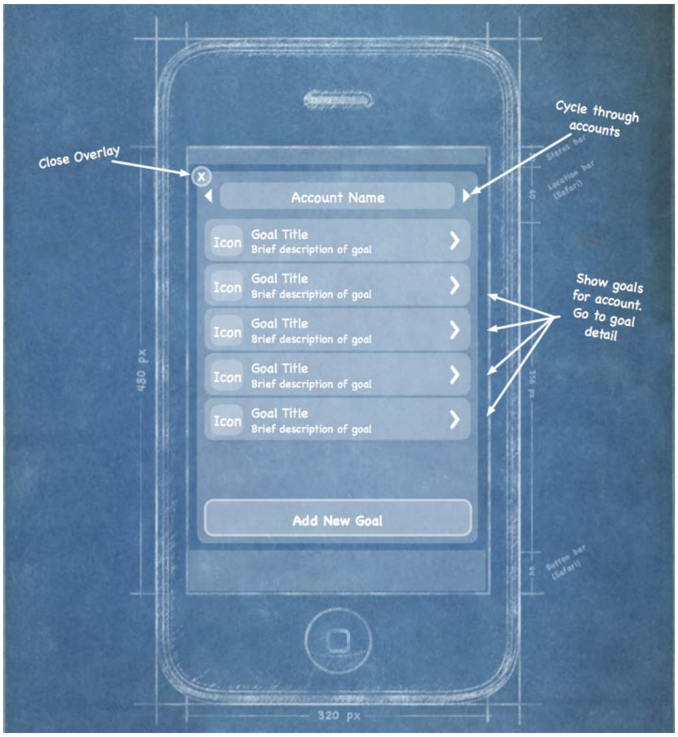

Close Overlay
Cycle through accounts
Account Name
Icon
Goal Title
Brief description of goal
Icon
Goal Title
Brief description of goal
Icon
Goal Title
Brief description of goal
Icon
Goal Title
Brief description of goal
Icon
Goal Title
Brief description of goal
Show goals for account. Go to goal detail
Add New Goal
480 px
320 px

Figure 7-10. Using annotations to indicate the desired interactions of the site or application

But the purpose of wireframes is not just to provide a visual for our site map; they also serve to separate layout from visual design, defining how the user will interact with the experience. How do we lay out our navigation? What visual or interaction metaphors will we use to evoke action? What are the best ways to communicate and show information in the assumed context of the user? These questions and many more are answered with wireframes.

Although I've found wireframes to be one of the most valuable information deliverables to communicate my vision for how a site or app will work, the challenge is that a diagram on a piece of paper doesn't go a long way toward describing how the interactions will work. Most common are what I call "in-place" interactions, or areas where the user can interact with an element without leaving the page. This can be done with Ajax or a little show/hide JavaScript. These interactions can include copious amounts of annotation, describing each content area in as much length as you can fit in the margins of the page, as shown in Figure 7-10.

At this point, I highly recommend that you get some feedback from either others on your project or my most trusted reviewer, my wife. Well, not *my* wife, but someone you know and trust—and the less technical, the better. Have her review your work as you iterate through ideas. Describe what problems you are trying to solve and ask her what she is thinking. It has been an invaluable process for me over the years, not specifically for the feedback I receive—though you do get the occasional strokes of genius—but for forcing me to communicate what I've done and verbalize my thoughts. You'd be surprised how often you get an idea in your head that you think is brilliant, but once you say it out loud, it just sounds absurd.

In mobile, it is this kind of feedback, using wireframes as your key deliverable, that turns good ideas into excellent mobile products.

Prototyping

As mentioned before, wireframes lack the capability to communicate more complex, often in-place, interactions of mobile experiences. This is where prototypes come in. Prototypes might sound like a scary (or costly) step in the process. Some view them as redundant or too time-consuming, preferring to jump in and start coding things. But as with wireframes, I've found that each product we've built out some sort of prototype has saved both time and money. The following sections discuss some ways to do some simple and fast mobile prototyping.

Paper prototypes

The most basic level we have is paper prototyping: taking our printed-out wireframes or even drawings of our interface, like the one shown in Figure 7-11, and putting them in front of people.

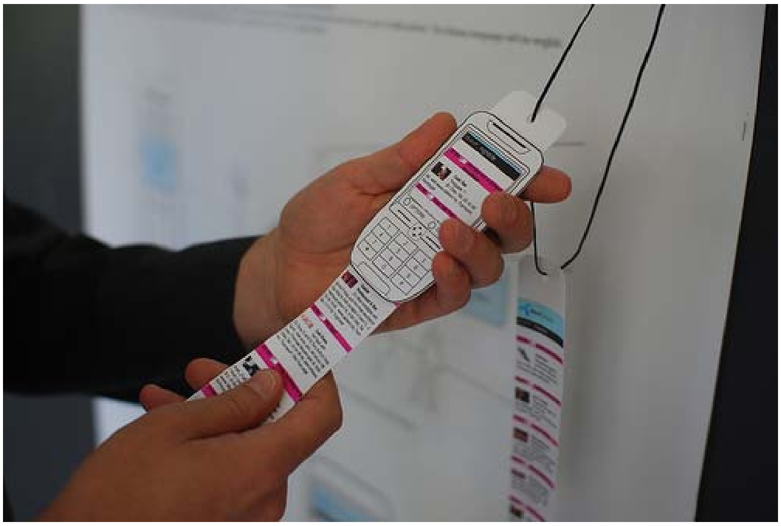


Figure 7-11. A paper prototype, where the interaction is nothing more than drawings on note cards

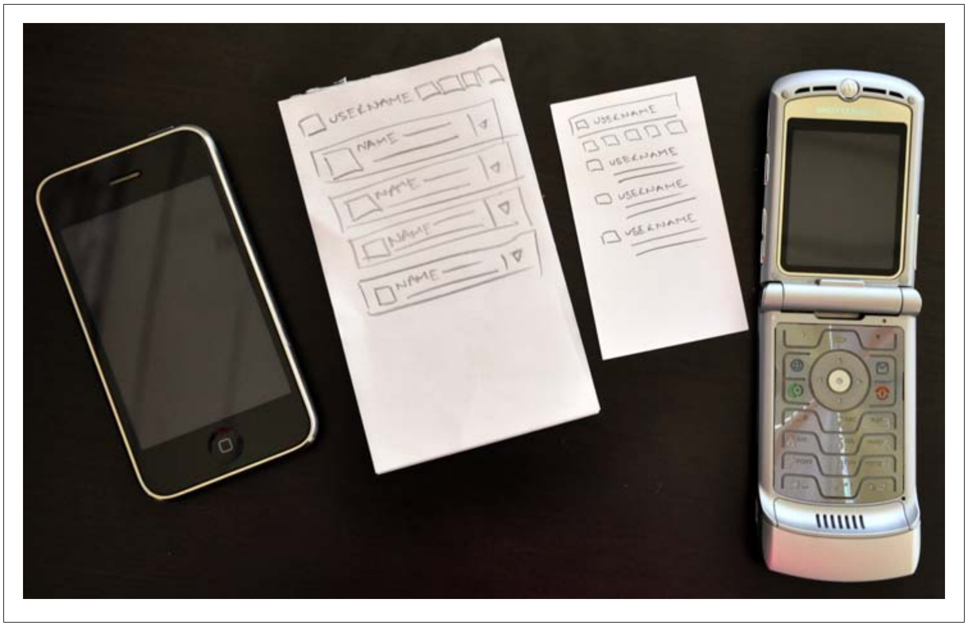


Figure 7-12. A touch interface paper prototype next to its smaller sibling

Create a basic script of tasks (hopefully based on either context or user input) and ask users to perform them, pointing to what they would do. You act as the device, changing the screens for them. For paper prototypes, I don't recommend using full sheets of paper; instead try using small blank note cards, and for lower-end devices, use business-card-sized paper (Figure 7-12). The size matters and you'll learn as much from how the user manages working with small media as you will what information is actually on it.

Context prototype

The next step is creating a context prototype (Figure 7-13). Take a higher-end device that enables you to load full-screen images on it. Take your wireframes or sketches and load them onto the device, sized to fill the device screen. Leave the office. Go for a walk down to your nearest café. Or get on a bus or a train. As you are traveling about, pull out your device and start looking your interface in the various contexts you find yourself currently in.

Pay particular attention to what you are thinking and your physical behavior while you are using your interface and then write it down. If you are brave and don't have strict nondisclosure issues, ask the people around you to use it, too. I wouldn't bother with timing interactions or sessions, but try to keep an eye on a clock to determine how long the average session is.

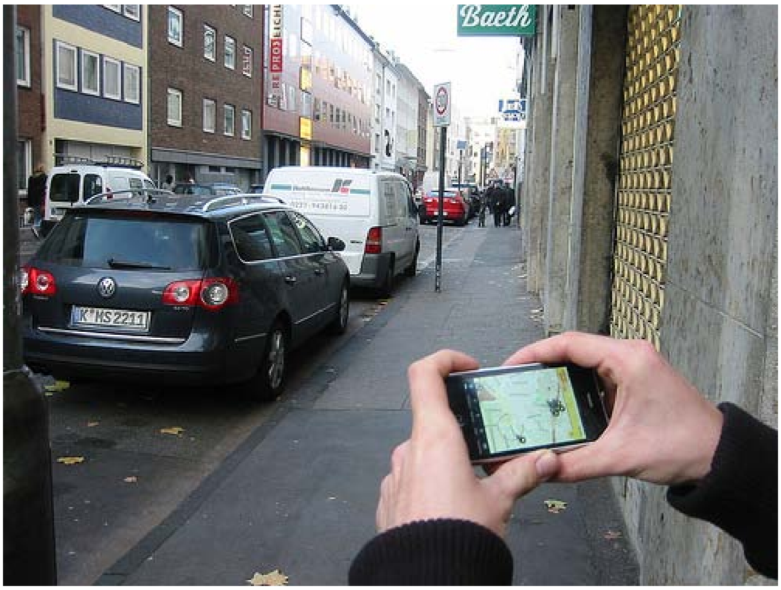


Figure 7-13. An example of a context prototype, or taking images loaded onto a device and testing them in the mobile context

HTML prototypes

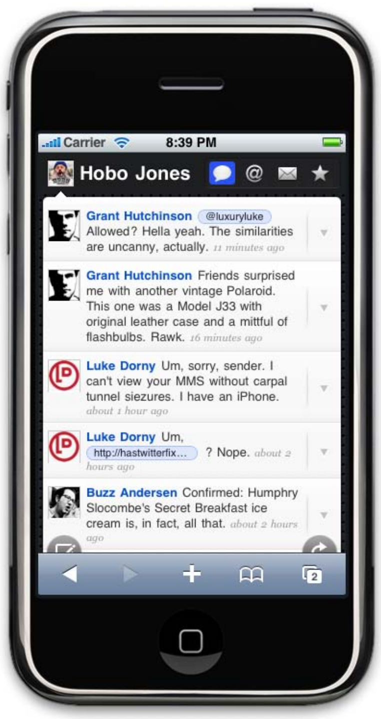
The third step is creating a lightweight, semifunctional static prototype using XHTML, CSS, and JavaScript, if available. This is a prototype that you can actually load onto a device and produce the nearest experience to the final product, but with static dummy content and data (Figure 7-14). It takes a little extra time, but it is worth the effort.

With a static XHTML prototype, you use all the device metaphors of navigation, you see how much content will really be displayed on screen (it is always less than you expect), and you have to deal with slow load times and network latency. In short, you will feel the same pains your user will go through.

Whatever route you wish to take, building a mobile prototype takes you one very big leap forward to creating a better mobile experience. I won't lie: it can feel like a thankless exercise at times, but your users will thank you in the long run by using your app.

Different Information Architecture for Different Devices

Depending on which devices you need to support, mobile wireframes can range from the very basic to the complex. On the higher-end devices with larger screens, we might be inclined to add more interactions, buttons, and other clutter to the screen, but this would be a mistake. Just because the user might have a more advanced phone, that



Grant Hutchinson @luxuryluke Allowed? Hella yeah. The similarities are uncanny, actually. 11 minutes ago

Grant Hutchinson Friends surprised me with another vintage Polaroid. This one was a Model J33 with original leather case and a mittful of flashbulbs. Rawk. 16 minutes ago

Luke Dorny Um, sorry, sender. I can't view your MMS without carpal tunnel siezures. I have an iPhone. about 1 hour ago

Luke Dorny Um, http://hastwitterfix... ? Nope. about 2 hours ago

Buzz Andersen Confirmed: Humphry Slocombe's Secret Breakfast ice cream is, in fact, all that. about 2 hours ago

Figure 7-14. An XHTML prototype that you can actually interact with on real mobile devices

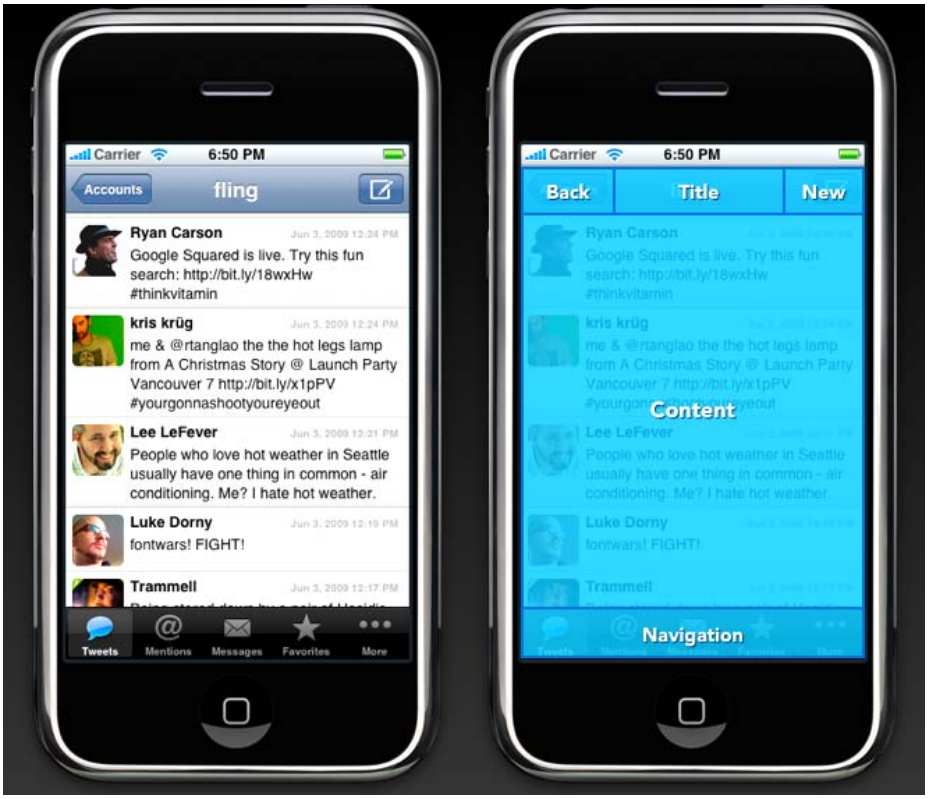
doesn't mean that he is giving you license to pack his screen with as much information as you can muster.

The motivations, goals, and how users will interact with a mobile experience are the same at the low end as they are on a high-end device. For the latter, you just have better tools to express the content. You can learn a lot from designing for the lower end first. The greatest challenge in creating valuable experiences is knowing when to lose what you don't need. You don't have a choice on lower-end devices—it must be simple. When designing for both, it is best to try and to keep your information architecture as close to each other as possible without sacrificing the user experience. They say that simple design is the hardest design, and this principle certainly is true when designing information architecture for mobile devices.

The Design Myth

A little secret about interactive design is that people don't respond to the visual aesthetic as much as you might think. What colors you use, whether you use square or rounded corners, or, gradients or flat backgrounds, helps build first impressions, but it doesn't do too much to improve the user's experience. Don't get me wrong: users appreciate good design, but they are quickly indifferent about the visual aesthetic and move almost immediately to the layout (information design), what things are called (taxonomy), the findability of content, and how intuitive it is to perform tasks. These are all facets of information architecture.

Just look at one of the top-selling iPhone Twitter applications, Tweetie, shown in Figure 7-15. Many consider Tweetie to be a "well-designed" application, but because it is built from the same API as all other iPhone applications, at first glance there is little that is actually visually distinctive between this and other applications. What makes this application "well designed" is how the content is applied to the context of the user—in other words, the mobile information architecture. In this example, the information design uses common layout metaphors, highlighted on the righthand side of Figure 7-15 to provide the user with familiar placement of common tasks, allowing the user to perform repetitive tasks common with most Twitter applications. The point is great information design is often mistaken for great visual design.



Ryan Carson Jun 3, 2009 12:24 PM
Google Squared is live. Try this fun search: http://bit.ly/18wxHw #thinkvitamin

kris krüg Jun 3, 2009 12:24 PM
me & @rtanglao the the hot legs lamp from A Christmas Story @ Launch Party Vancouver 7 http://bit.ly/x1pPV #yourgonnashootyoureyeout

Lee LeFever Jun 3, 2009 12:21 PM
People who love hot weather in Seattle usually have one thing in common - air conditioning. Me? I hate hot weather.

Luke Dorny Jun 3, 2009 12:19 PM
fontwars! FIGHT!

Trammell Jun 3, 2009 12:17 PM

Figure 7-15. Comparing visual design to information design of the iPhone application Tweetie

Most non–information architects almost always do information architecture in some form or another; often, they don't even know they are doing it. They might do a few wireframes, or maybe a site map. Sometimes designers will jump in and incorporate information architecture deliverables directly into their designs. By not focusing on the information architecture exclusively from the start, you risk confusing your disciplines, your deliverables, and ultimately your direction. The more time you spend focusing on just your information architecture, the faster and less costly your project will be.

CHAPTER 8

Mobile Design

When building mobile experiences, it is impossible to create a great experience without three ingredients: context, information architecture, and visual design. This chapter focuses on the latter ingredient of the recipe. The visual design of your experience is the direct representation of everything underneath; it is the first impression the user will have. A great design gives the user high expectations of your site or application; a poor design leads to lower expectations.

Users' expectations translate to value and trust. In the mobile space, where content is often "free" (they still need to pay for data charges), users often have low expectations, due to the limitations of the canvas. Confusing site structures and slow download speeds reinforce those initial low expectations. In the downloadable application space, where application design can be much more robust, users must purchase applications almost sight unseen. For example, they may see just a small screenshot of your application or game. But if the application doesn't meet the higher expectations of the design, your application downloads will drop like a stone. The design, that first impression, determines right from the start if the user will spend five seconds, five minutes, or five hours with your product.

This leads us to the most significant challenge in mobile design: creativity. You can't always be as creative as you want to be. Many devices just can't support complex designs for every channel; for example, on many lower-end devices, the mobile web experience may just be a list of links. But every device has the capability to create a best-in-device experience; it just depends on how you take advantage of the application medium and context that you plan to use.

On computers, there is a strategy called "lowest common denominator": in order to reach the widest possible number of platforms, you create a product that works on the most common architectural components on all platforms (see Figure 8-1). Well, in computers, where you may have under a dozen different platforms, this is a great concept, but in mobile development, where you might be dealing with hundreds of different devices, it becomes a necessity.

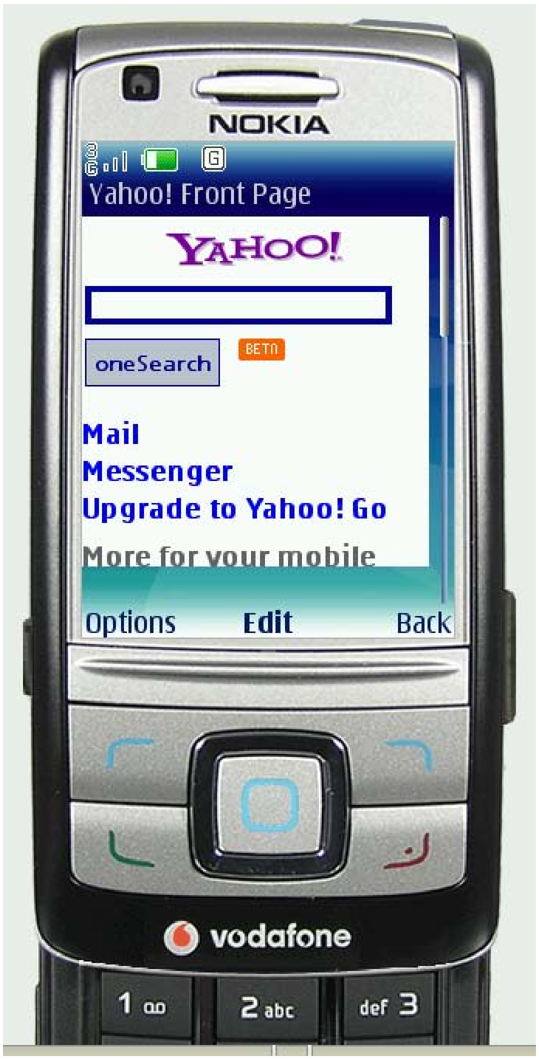


Figure 8-1. A lowest-common-denominator design

Typically, mobile design starts with the lowest common denominator. As a designer, you ask yourself, "How do I visually express this content across the most possible devices?" You start with the most basic of designs, catering to the limitations of the device. You try to pepper in some nice-looking elements until you've reached the extent that the device platform can tolerate. You are left with a Frankenstein-like design that only your mother could love.

Interpreting Design

Mobile design reminds me of a 25-year veteran graphic designer friend of mine. His days were spent creating print designs and advertisements, defining precisely what each design would be, down to the picas and points. His method of design meant creating a vision for how to communicate information or ideas in his head, and then duplicating that on the printed page. In his mind, it wasn't right unless it was exactly like his original vision.

I offered to help him with his website; looking back on this, I realize that it probably was not the wisest decision. He spent months obsessing about exactly how his site looked. I would try to explain to him that the Web isn't about creating precise experiences, like you do in print. I explained that my method of communicating information was to structure it with design, use design elements to enhance the information, and enable the user to interpret it. It is important to remember that every experience is unique. That experience depends on the user's screen size, web browser, text settings, the speed of his computer, and his connection to the Internet. There are simply too many variables for us to try to "control" the design completely.

This frustrated the veteran graphic designer. He could not look at my interpreted designs without trying to precisely position every element on the page. And if I resized the browser window, he would get angry.

I frequently think of this project when working with mobile designs. It isn't that either one of us was right or wrong. That is the great thing about design—it is completely subjective, giving designers plenty of things to argue about. We just had two very different ways of using design to express information, based on the fact that we came from different media. He wanted it to treat the design precisely, to recreate his vision exactly. I wanted it to be flexible, catering to the unknown variables of the medium. The reality is that we probably should have met someplace in the middle.

Mobile design isn't that different. Precise designs might look better, but they can be brutal to implement. More flexible designs might not be much to look at, but they work for the most users, or the lowest common denominator. But more than that, our backgrounds and our training can actually get in the way of creating the best design for the medium. We like to apply the same rules to whatever the problem in front of us might be. In mobile design, you interpret what you know about good design and translate it to this new medium that is both technologically precise and at times incredibly unforgiving, and you provide the design with the flexibility to present information the way you envision on a number of different devices.

In the end, the graphic designer and I scrapped the work, and he provided me his pica-perfect designs as giant images, which I turned into a series of massive image maps. To this day, I do not work with graphic designers on web or mobile projects.

The Mobile Design Tent-Pole

In Hollywood, executives like to use the term “tent-pole” to describe their movies and television shows. The term has dual meanings: one is business, and the other creative. The business goal of a tent-pole production is to support or prop up the losses from other productions. However, to create a tent-pole production, the creators involved must make an artistic work that they know will appeal to the largest possible audience, providing something for everyone. You probably know tent-pole movies as “blockbusters”; in television, they are known as “anchor” shows.

Trying to reach the widest possible audience poses a problem. Hollywood is learning with great pains that with so many entertainment options and with today’s audience being so hard to reach through traditional advertising channels, tent-pole productions are failing. As the number of social niches increases, it becomes difficult to satisfy the specific tastes of each social group. What one group finds hysterically funny, several other groups might find offensive. Today, tent-pole productions often come off as bland and half-hearted, failing to appease anyone.

One of the most interesting examples of how the tide turned in entertainment is with the animated films of Disney versus those of Pixar. For years, Disney produced tent-pole family fare quite successfully. But as competition increased, notably from Pixar, Disney films would spend millions to create stale and dated films, losing audiences and revenue. Meanwhile, Pixar found that their movies could be successful by avoiding the traditional storytelling formats of animated film, which Disney essentially defined. Instead, Pixar based their stories around specific emotional themes and was able to connect with audiences of all ages, in multiple cultures and across multiple niches.

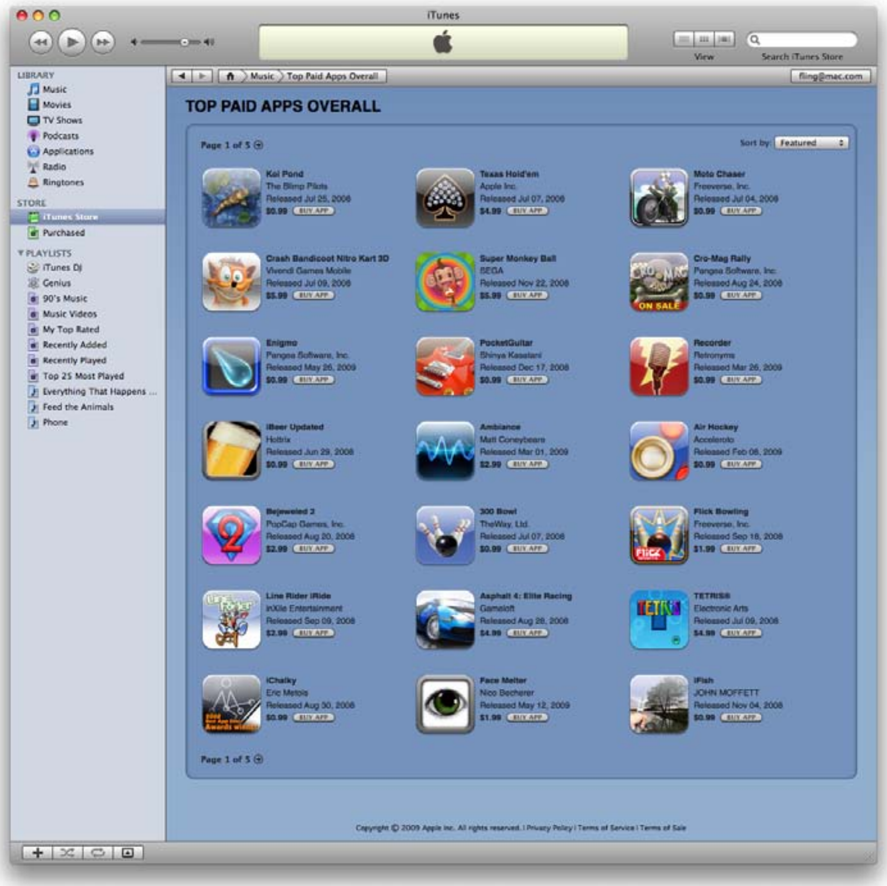
In 2006, Disney acquired Pixar, making its top executives the new leaders of all Disney creative projects. Although Disney technically acquired Pixar, I’ve always looked at it as the other way around. Disney realized that it needed to be more Pixar and less Disney in order to grow and adapt to today’s changing audiences and niches. This is something that Pixar was doing correctly.

Back in the world of mobile design, the de facto strategy is to create tent-pole products. Like the old days of Disney, the strategy is to sink millions into creating tent-pole products, or products that support the largest number of devices that no one will ever use. They are creatively stale, they lack inspiration, and they simply exist with no meaningful purpose to the user. They make the same mistake Disney made, thinking that it could simply put something on the market that might not be the best quality, but because it carried the Disney name, people would buy it.

To have a successful mobile design, you have to think like Pixar. Find that emotional connection, that fundamental need that serves many audiences, many cultures, and many niches and design experiences. Too often, designers simply echo the visual trends of the day, mimicking the inspiration of others—I’m certainly guilty of it. But with mobile design, once you find that essential thing, that chewy nougat we call “context” that lives at the center of your product, then you will find ample inspiration of your own to start creating designs that translate not only across multiple devices, but across multiple media.

Sure, there are countless examples of poorly designed mobile products that are considered a success. You only need to look as far as the nearest mobile app store to find them. This is because of the sight unseen nature of mobile commerce. Traditionally, you couldn't demo—or in some cases even see—screenshots of a game or mobile application before you bought it. You simply had to purchase it and find out if it was any good.

Apple's App Store quickly changed that. You can clearly see that the best-selling games and applications for the iPhone are the ones with the best designs (Figure 8-2).



TOP PAID APPS OVERALL

Page 1 of 5

Koi Pond
The Blimp Pilots
Released Jul 25, 2008
$0.99

Texas Hold'em
Apple Inc.
Released Jul 07, 2008
$4.99

Moto Chaser
Freeverse, Inc.
Released Jul 04, 2008
$0.99

Crash Bandicoot Nitro Kart 3D
Vivendi Games Mobile
Released Jul 09, 2008
$5.99

Super Monkey Ball
SEGA
Released Nov 22, 2008
$5.99

Cro-Mag Rally
Pangea Software, Inc.
Released Aug 24, 2008
$0.99

Enigmo
Pangea Software, Inc.
Released May 26, 2009
$0.99

PocketGuitar
Shinya Kasatani
Released Dec 17, 2008
$0.99

Recorder
Retronyms
Released Mar 26, 2009
$0.99

iBeer Updated
Hottrix
Released Jun 29, 2008
$0.99

Ambiance
Matt Coneybeare
Released Mar 01, 2009
$2.99

Air Hockey
Acceleroto
Released Feb 08, 2009
$0.99

Bejeweled 2
PopCap Games, Inc.
Released Aug 20, 2008
$2.99

300 Bowl
TheWay, Ltd.
Released Jul 07, 2008
$0.99

Flick Bowling
Freeverse, Inc.
Released Sep 18, 2008
$1.99

Line Rider iRide
inXile Entertainment
Released Sep 09, 2008
$2.99

Asphalt 4: Elite Racing
Gameloft
Released Aug 28, 2008
$4.99

TETRIS®
Electronic Arts
Released Jul 09, 2008
$4.99

iChalky
Eric Metois
Released Aug 30, 2008
$0.99

Face Melter
Nico Becherer
Released May 12, 2009
$1.99

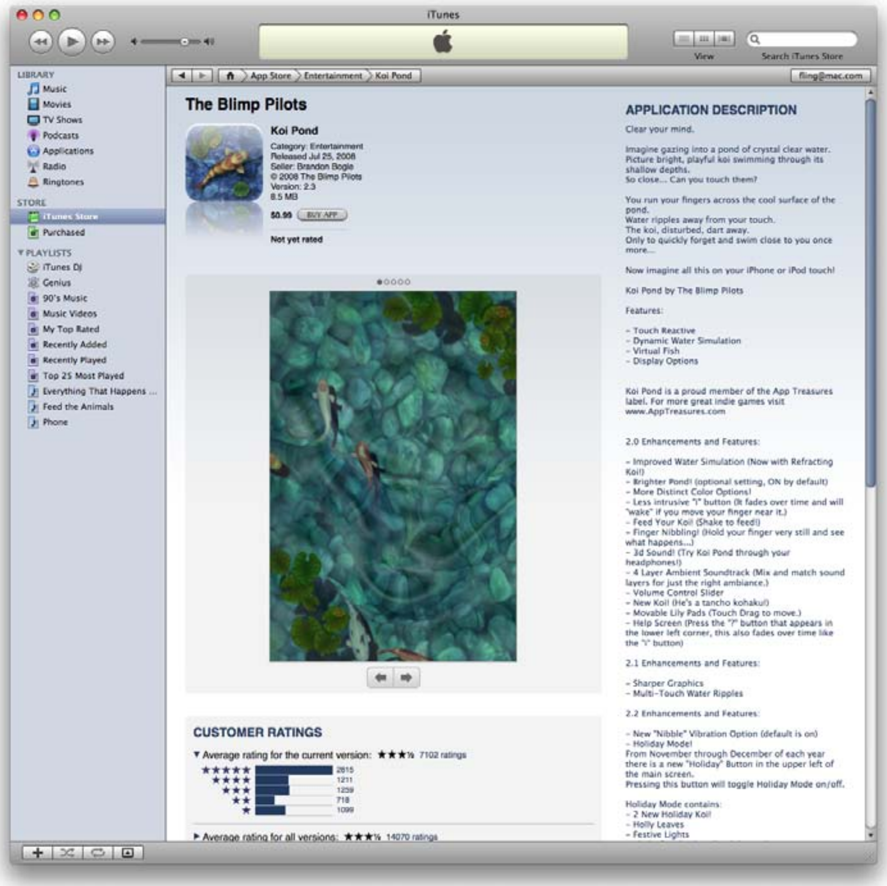
iFish
JOHN MOFFETT
Released Nov 04, 2008
$0.99

Page 1 of 5

Copyright © 2009 Apple Inc. All rights reserved. | Privacy Policy | Terms of Service | Terms of Sale

Figure 8-2. The app icon design greatly influences the user's expectation of quality

Users look at multiple screenshots (Figure 8-3), read the user reviews, and judge the product based on the quality of its icon and of the screenshots before they buy.



The Blimp Pilots

Koi Pond
Category: Entertainment
Released Jul 25, 2008
Seller: Brandon Bogle
© 2008 The Blimp Pilots
Version: 2.3
8.5 MB
$0.99

Not yet rated

CUSTOMER RATINGS

▼ Average rating for the current version: ★★★½ 7102 ratings

★★★★★	2815
★★★★	1211
★★★	1259
★★	718
★	1099

▶ Average rating for all versions: ★★★½ 14070 ratings

APPLICATION DESCRIPTION

Clear your mind.

Imagine gazing into a pond of crystal clear water.
Picture bright, playful koi swimming through its shallow depths.
So close... Can you touch them?

You run your fingers across the cool surface of the pond.
Water ripples away from your touch.
The koi, disturbed, dart away.
Only to quickly forget and swim close to you once more...

Now imagine all this on your iPhone or iPod touch!

Koi Pond by The Blimp Pilots

Features:

- Touch Reactive
- Dynamic Water Simulation
- Virtual Fish
- Display Options

Koi Pond is a proud member of the App Treasures label. For more great indie games visit www.AppTreasures.com

2.0 Enhancements and Features:

- Improved Water Simulation (Now with Refracting Koi!)
- Brighter Pond! (optional setting, ON by default)
- More Distinct Color Options!
- Less intrusive "i" button (It fades over time and will "wake" if you move your finger near it.)
- Feed Your Koi! (Shake to feed!)
- Finger Nibbling! (Hold your finger very still and see what happens...)
- 3d Sound! (Try Koi Pond through your headphones!)
- 4 Layer Ambient Soundtrack (Mix and match sound layers for just the right ambiance.)
- Volume Control Slider
- New Koi! (He's a tancho kohaku!)
- Movable Lily Pads (Touch Drag to move.)
- Help Screen (Press the "?" button that appears in the lower left corner, this also fades over time like the "i" button)

2.1 Enhancements and Features:

- Sharper Graphics
- Multi-Touch Water Ripples

2.2 Enhancements and Features:

- New "Nibble" Vibration Option (default is on)
- Holiday Mode!
From November through December of each year there is a new "Holiday" Button in the upper left of the main screen.
Pressing this button will toggle Holiday Mode on/off.

Holiday Mode contains:
- 2 New Holiday Koi!
- Holly Leaves
- Festive Lights

Figure 8-3. Users are able to determine the quality of the app, largely influenced by the design, before they make a purchase

The Apple App Store is proving everyday that mobile design doesn't have to start with tent-pole lowest-common-denominator products—it can instead start with providing the best possible experience and tailoring that experience to the market that wants it most.

Designing for the Best Possible Experience

When the first iPhone came out, I got in a lot of trouble from my web and mobile peers for publicly saying, "The iPhone is the only mobile device that matters right now." They would argue, "What about ABC or XYZ platforms?" My response was that those are important, but the iPhone provides the best possible experience and that is where consumers will go. Since those days, we've seen the iPhone shatter just about every record in mobile devices, becoming one of the best-selling phones ever and one of the most used mobile browsers in the world—two-thirds of mobile browsing in the U.S. comes from an iPhone or an iPod touch, not to mention that more than a billion mobile applications have been sold for these devices in under a year.

Recently, I was speaking at a conference where I ran into one of my peers, who questioned my premise that the iPhone was the most important device in mobile. He came up to me, and the first thing he said was, "I remember you telling me ages ago that the iPhone is the only device that mattered, and I didn't believe you. And here we are today focusing our business on the iPhone." It was an odd (and rare) reverse I-told-you-so moment. Here was this seasoned mobile guy telling me that his instincts had been wrong and my instincts had been right. I thought it must have been hard for him to go against his instincts and shift not just his thinking but his entire business toward supporting one popular device.

The lesson here is that although it may defy your business instincts to focus your product on just one device, in mobile development, the risks and costs of creating that tent-pole product are just too high. This lesson is so easily seen through bad or just plain uninspired mobile design. Asking creative people to create uninspiring work is a fast track to mediocrity.

Here is a design solution: design for the best possible experience. Actually, don't just design for it: focus on creating the best possible experience with unwavering passion and commitment. Iterate, tweak, and fine-tune until you get it right. Anything less is simply unacceptable. Do not get hindered by the constraints of the technology. Phrases like "lowest common denominator" cannot be part of the designer's vocabulary.

Your design—no, your work of art—should serve as the shining example of what the experience should be, not what it can be. Trying to create a mobile design in the context of the device constraints isn't where you start; it is where you should end.

I think one of the greatest mistakes we in the mobile community make is being unwilling to or feeling incapable of thinking forward. The tendency to frame solutions in the past (past devices, past standards) applies only to those low-quality, something-for-everyone-but-getting-nothing tent-pole products. Great designs are not unlike great leaps forward in innovation. They come from shedding the baggage regarding how things are done and focus on giving people what they want or what they need.

The Elements of Mobile Design

As I wrote this chapter, I struggled with how to describe how to do design. I personally believe that good design requires three abilities: the first is a natural gift for being able to see visually how something should look that produces a desired emotion with the target audience. The second is the ability to manifest that vision into something for others to see, use, or participate in. The third is knowing how to utilize the medium to achieve your design goals.

Although I can't teach you how to remap your brain (at least not in the scope of this book) to be a designer, I can teach you how to at least think like a designer. Doing this involves knowing the six elements of mobile design that you need to consider, starting with the context and layering in visual elements or laying out content to achieve the design goal. Then, you need to understand how to use the specific tools to create mobile design, and finally, you need to understand the specific design considerations of the mobile medium.

Context

I discussed context in Chapter 4. I won't belabor the point except to say that context is core to the mobile experience. As the designer, it is your job to make sure that the user can figure out how to address context using your app. Make sure you do your homework to answer the following questions:

- Who are the users? What do you know about them? What type of behavior can you assume or predict about the users?
- What is happening? What are the circumstances in which the users will best absorb the content you intend to present?
- When will they interact? Are they at home and have large amounts of time? Are they at work where they have short periods of time? Will they have idle periods of time while waiting for a train, for example?
- Where are the users? Are they in a public space or a private space? Are they inside or outside? Is it day or is it night?
- Why will they use your app? What value will they gain from your content or services in their present situation?
- How are they using their mobile device? Is it held in their hand or in their pocket? How are they holding it? Open or closed? Portrait or landscape?

The answers to these questions will greatly affect the course of your design. Treat these questions as a checklist to your design from start to finish. They can provide not only great inspiration for design challenges, but justification for your design decisions later.

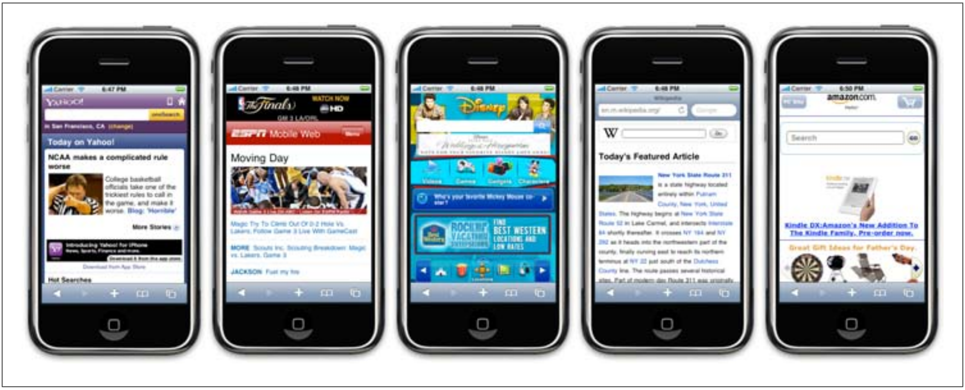
Message

Another design element is your message, or what you are trying to say about your site or application visually. One might also call it the "branding," although I see branding and messaging as two different things. Your message is the overall mental impression you create explicitly through visual design. I like to think of it as the holistic or at times instinctual reaction someone will have to your design. If you take a step back, and look at a design from a distance, what is your impression? Or conversely, look at a design for 30 seconds, and then put it down. What words would you use to describe the experience?

Branding shouldn't be confused with messaging. Branding is the impression your company name and logo gives—essentially, your reputation. Branding serves to reinforce the message with authority, not deliver it. In mobile, the opportunities for branding are limited, but the need for messaging is great. With such limited real estate, the users don't care about your brand, but they will care about the messaging, asking themselves questions like, "What can this do for me?" or "Why is this important to me?"

Your approach to the design will define that message and create expectations. A sparse, minimalist design with lots of whitespace will tell the user to expect a focus on content. A "heavy" design with use of dark colors and lots of graphics will tell the user to expect something more immersive.

For example, hold the book away from you and look at each of the designs in Figure 8-4; try not to focus too heavily on the content. What do each of these designs "say" to you?



Yahoo!
Today on Yahoo!
NCAA makes a complicated rule worse
ESPN Mobile Web
Moving Day
Disney
Today's Featured Article
amazon.com
Kindle DX: Amazon's New Addition To The Kindle Family. Pre-order now.

Figure 8-4. *What is the message for each of these designs?*

Which of the following designs provide a message? What do they say to you?

Yahoo!

Yahoo! sort of delivers a message. This app provides a clean interface, putting a focus on search and location, using color to separate it from the news content. But

I'm not exactly sure what it is saying. Words you might use to describe the message are crisp, clean, and sharp.

ESPN
: The ESPN site clearly is missing a message. It is heavily text-based, trying to put a lot of content above the fold, but doesn't exactly deliver a message of any kind. If you took out the ESPN logo, you likely would have indifferent expectations of this site; it could be about anything, as the design doesn't help set expectations for the user in any way. Words you might use to describe the message: bold, cluttered, and content-heavy.

Disney
: Disney creates a message with its design. It gives you a lot to look at—probably too much—but it clearly tries to say that the company is about characters for a younger audience. Words you might use to describe the message: bold, busy, and disorienting.

Wikipedia
: The Wikipedia design clearly establishes a message. With a prominent search and text-heavy layout featuring an article, you know what you are getting with this design. Words you might use to describe the message: clean, minimal, and text-heavy.

Amazon
: Amazon sort of creates a message. Although there are some wasted opportunities above the fold with the odd ad placement, you can see that it is mostly about products (which is improved even more if you scroll down). Words you might use to describe the message: minimal but messy, product-heavy, and disorienting.

The words you might use to describe these designs might be completely different than mine—thus the beauty and the curse of visual design. The important thing isn't my opinion—it is the opinion of your user. Does the design convey the right message to your user in the right context? If you aren't sure, it might be a good time to find out.

Look and Feel

The concept of "look and feel" is an odd one, being subjective and hard to define. Typically, look and feel is used to describe appearance, as in "I want a clean look and feel" or "I want a usable look and feel." The problem is: as a mobile designer, what does it mean? And how is that different than messaging?

I think of look and feel in a literal sense, as something real and tactile that the users can "look" at, then "feel"—something they can touch or interact with. Look and feel is used to evoke action—how the user will use an interface. Messaging is holistic, as the expectation the users will have about how you will address their context. It is easy to confuse the two, because "feel" can be interpreted to mean our emotional reaction to design and the role of messaging.

I prefer to keep the concept of look and feel grounded in a tangible design, something I can clearly describe and show to users. I often find myself explaining the look and feel with the word "because," with a cause-and-effect rationale for design decisions, as in "The user will press this button because..." or "The user will go to this screen because..." followed by a reason why a button or control is designed a certain way.

Establishing a look and feel usually comes from wherever design inspiration comes from. However, your personal inspiration can be a hard thing to justify. Therefore we have "design patterns," or documented solutions to design problems, sometimes referred to as style guides. On large mobile projects or in companies with multiple designers, a style guide or pattern library is crucial, maintaining consistency in the look and feel and reducing the need for each design decision to be justified. For example, in Figure 8-5 you can see the site Pattern Tap, which is a visual collection of many user interface patterns meant for websites and web applications, but there is no reason why it can't serve as inspiration for your mobile projects as well.

In Figure 8-6 you can see an example of a mobile design pattern at the Design4Mobile pattern library.

Although a lot of elements go into making Apple's App Store successful, the most important design element is how it looks and feels. Apple includes a robust user interface tool that enables developers to use prebuilt components, supported with detailed Human Interface Guidelines (or HIG) of how to use them, similar to a pattern library. This means that a developer can just sit down and create an iPhone application that looks like it came from Apple in a matter of minutes. During the App Store submission process, Apple then ensures that the developer uses these tools correctly according to the HIG.

The look and feel can either be consistent with the stock user interface elements that Apple provides; they can be customized, often retaining the "spirit" of Apple's original design; or an entirely new look and feel can be defined—this approach is often used for immersive experiences.

The stock user experience that Apple provides is a great example of how look and feel works to supporting messaging. For the end user, the design sends a clear message: by using the same visual interface metaphors that Apple uses throughout the iPhone, I can expect the action, or how this control will behave, but I can also expect the same level of quality. This invokes the message of trust and quality in the application and in the platform as a whole. Apple isn't the first to use this shared look and feel model in mobile—in fact, it is incredibly common with most smartphone platforms—but they are surely making it incredibly successful, with a massive catalog of apps and the sales to support it.

My advice to would-be mobile designers is be creative and remember the context. Like in the early days of the Web, people tend to be skeptical about mobile experiences. The modal context of the user—in this case, what device he is using—should be considered during the design, as it will help to establish the user's expectations of the experience.

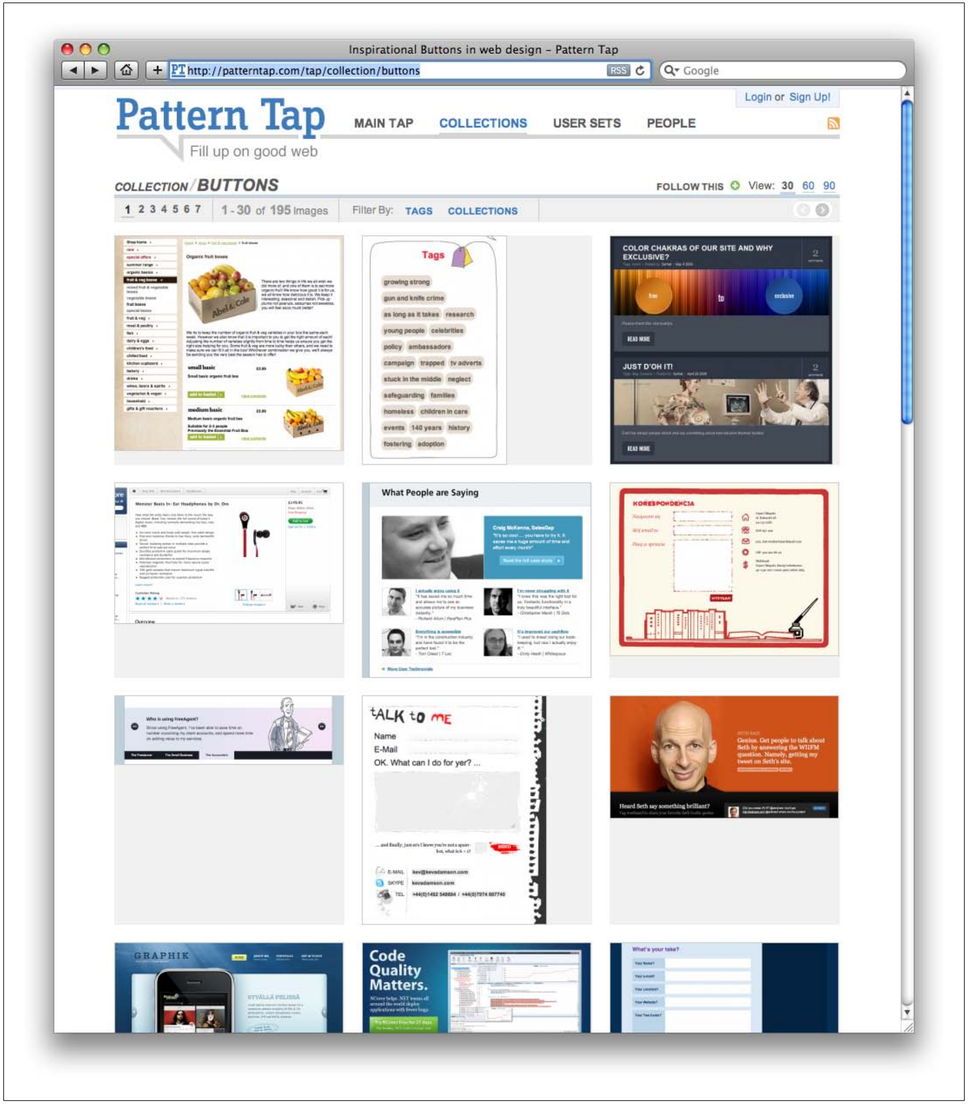
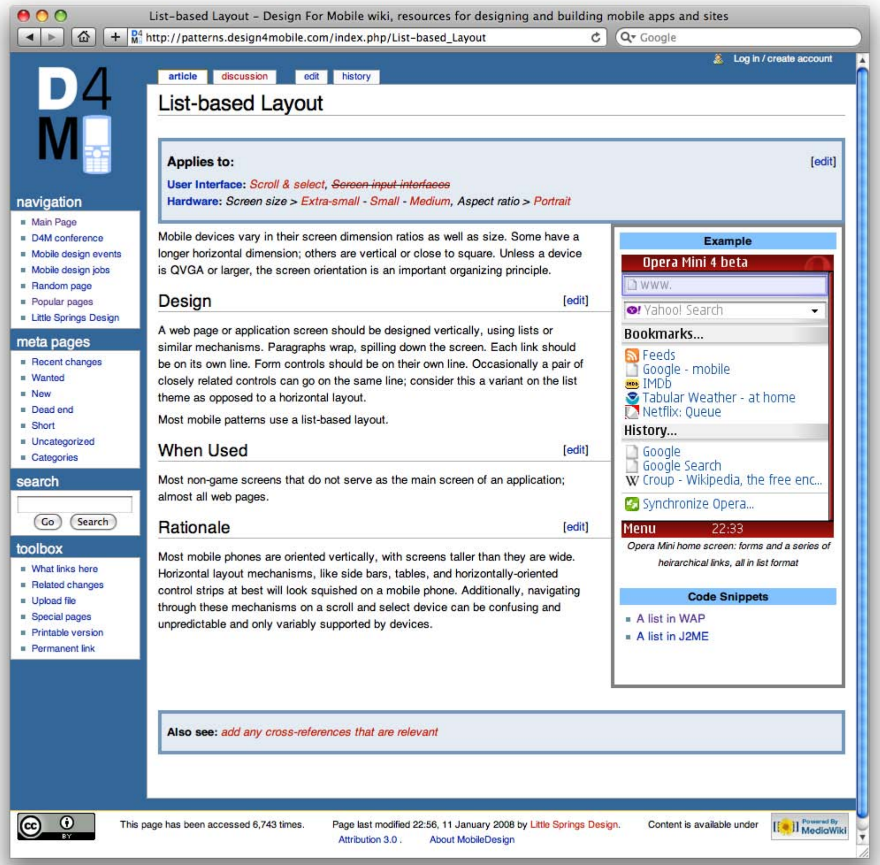


Figure 8-5. Pattern Tap shows a number of user interface patterns that help to establish look and feel

You can leverage this trust to your advantage, or you can strike out on your own and forge your own metaphors. As long as you know your users and the preferred mode of context, you can create a look and feel that is right for them.


List-based Layout – Design For Mobile wiki, resources for designing and building mobile apps and sites

http://patterns.design4mobile.com/index.php/List-based_Layout

Log in / create account

article | discussion | edit | history

List-based Layout

Applies to: [edit]

User Interface: *Scroll & select*, ~~*Screen input interfaces*~~

Hardware: *Screen size > Extra-small - Small - Medium, Aspect ratio > Portrait*

Mobile devices vary in their screen dimension ratios as well as size. Some have a longer horizontal dimension; others are vertical or close to square. Unless a device is QVGA or larger, the screen orientation is an important organizing principle.

Design [edit]

A web page or application screen should be designed vertically, using lists or similar mechanisms. Paragraphs wrap, spilling down the screen. Each link should be on its own line. Form controls should be on their own line. Occasionally a pair of closely related controls can go on the same line; consider this a variant on the list theme as opposed to a horizontal layout.

Most mobile patterns use a list-based layout.

When Used [edit]

Most non-game screens that do not serve as the main screen of an application; almost all web pages.

Rationale [edit]

Most mobile phones are oriented vertically, with screens taller than they are wide. Horizontal layout mechanisms, like side bars, tables, and horizontally-oriented control strips at best will look squished on a mobile phone. Additionally, navigating through these mechanisms on a scroll and select device can be confusing and unpredictable and only variably supported by devices.

Example

Opera Mini home screen: forms and a series of heirarchical links, all in list format

Code Snippets

- A list in WAP
- A list in J2ME

Also see: *add any cross-references that are relevant*

This page has been accessed 6,743 times. Page last modified 22:56, 11 January 2008 by Little Springs Design. Content is available under Attribution 3.0 . About MobileDesign

Figure 8-6. Design4Mobile provides a list of common mobile design patterns

Layout

Layout is an important design element, because it is how the user will visually process the page, but the structural and visual components of layout often get merged together, creating confusion and making your design more difficult to produce.

The first time layout should rear its head is during information architecture. In fact, I prefer to make about 90 percent of my layout decisions during the information architecture period. I ask myself questions like: where should the navigation go on the page or screen? What kind of navigation type should I use? Should I use tabs or a list? What about a sidebar for larger screens? All of these should be answered when defining the information architecture and before you begin to design.

Why define the layout before the mobile design? Design is just too subjective of an issue. If you are creating a design for anyone but yourself, chances are good that there will be multiple loosely-based-on-experience opinions that will be offered and debated. There is no right answer—only opinions and gut instincts. Plus, in corporate environments you have internal politics you have to consider, where the design opinions of the CEO or Chief Marketing Officer (CMO) might influence a design direction more than, say, the Creative Director or Design Director.

By defining design elements like layout prior to actually applying the look and feel, you can separate the discussion. As a self-taught designer, I started out in this business making designs for my own projects. I could just put pen to paper and tweak it to my heart's content. If I wanted to radically change the layout, I could. When I started my mobile design career with my first mobile company more than a decade ago, I realized that this approach didn't work. The majority of comments that reviewers would make were about the layout. They focused on the headers, the navigation, the footer, or how content blocks are laid out, and so on. But their feedback got muddied with the "look and feel, the colors, and other design elements."

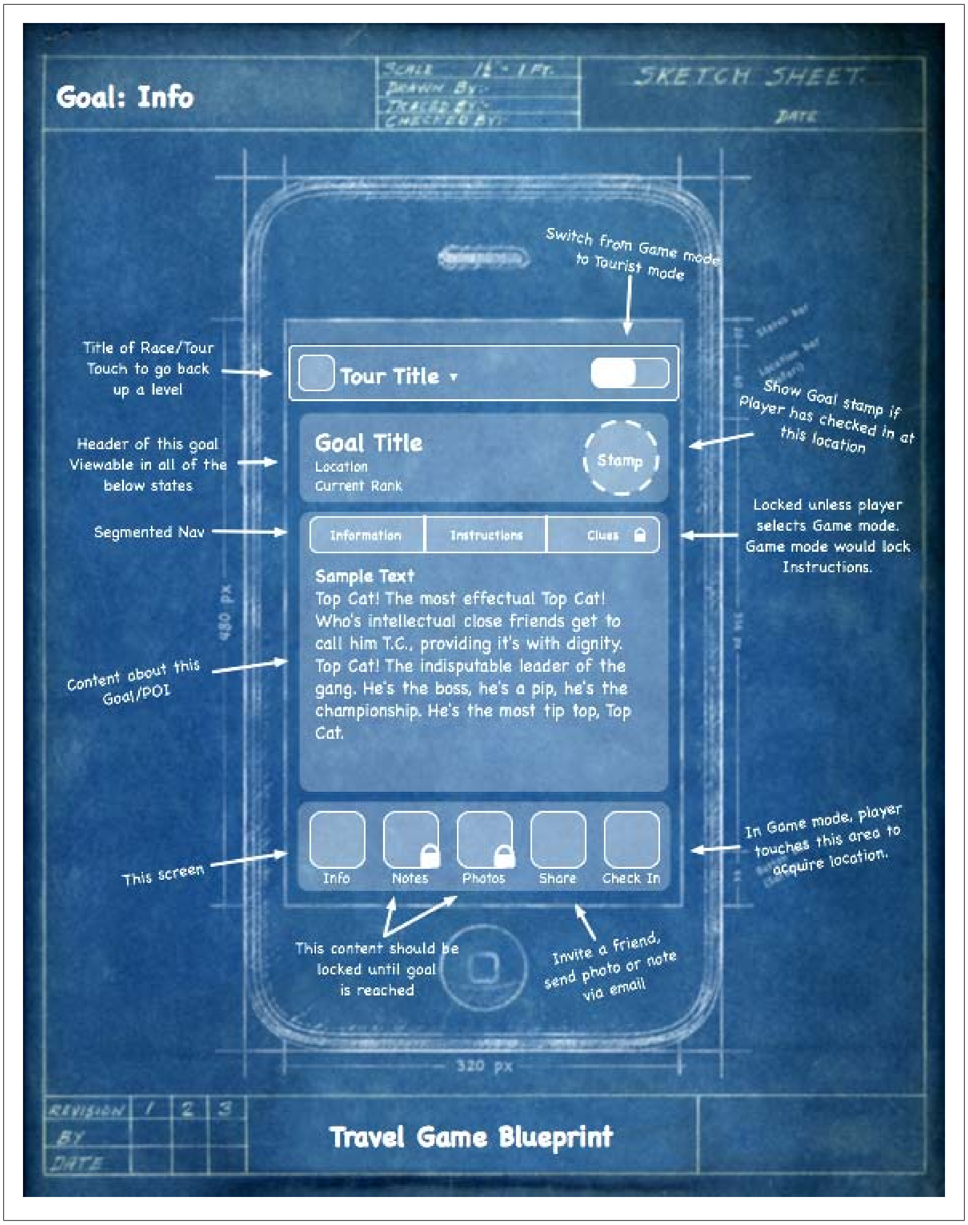
Reviewers do make remarks like "I like the navigation list, but can you make it look more raised?" Most designers don't hear that; they hear "The navigation isn't right, do it again." But, with this kind of feedback, there are two important pieces of information about different types of design. First, there is confirmation that the navigation and layout are correct. Second, there is a question about the "look and feel." Because designers hear "Do it again," they typically redo the layout, even though it was actually fine.

Creating mobile designs in an environment with multiple reviewers is all about getting the right feedback at the right time. Your job is to create a manifestation of a shared vision. Layout is one of the elements you can present early on and discuss independently. People confuse the quality and fidelity of your deliverables as design. By keeping it basic, you don't risk having reviewers confuse professionalism with design.

The irony is that as I become more adept at defining layouts, I make them of increasingly lower fidelity. For example, when I show my mobile design layouts as wireframes during the information architecture phase, I intentionally present them on blueprint paper, using handwriting fonts for my annotations (Figure 8-7). It also helps to say that this is not a design, it is a layout, so please give me feedback on the layout.

Different layouts for different devices

The second part of layout design is how to visually represent content. In mobile design, the primary content element you deal with the is navigation. Whether you are designing a site or app, you need to provide users with methods of performing tasks, navigating to other pages, or reading and interacting with content. This can vary, depending on the devices you support.


Goal: Info
SCALE 1½" = 1 FT.
DRAWN BY:
TRACED BY:
CHECKED BY:
SKETCH SHEET
DATE
Switch from Game mode to Tourist mode
Title of Race/Tour Touch to go back up a level
Tour Title ▾
Show Goal stamp if Player has checked in at this location
Header of this goal Viewable in all of the below states
Goal Title
Location
Current Rank
Stamp
Segmented Nav
Information
Instructions
Clues
Locked unless player selects Game mode. Game mode would lock Instructions.
Content about this Goal/POI
In Game mode, player touches this area to acquire location.
This screen
Info
Notes
Photos
Share
Check In
This content should be locked until goal is reached
Invite a friend, send photo or note via email
480 px
320 px
REVISION 1 2 3
BY
DATE
Travel Game Blueprint

Sample Text
Top Cat! The most effectual Top Cat! Who's intellectual close friends get to call him T.C., providing it's with dignity. Top Cat! The indisputable leader of the gang. He's the boss, he's a pip, he's the championship. He's the most tip top, Top Cat.

Figure 8-7. Using a low-fidelity wireframe to define the layout design element before visual design begins

There are two distinct types of navigation layouts for mobile devices: touch and scroll. With touch, you literally point to where you want to go; therefore, navigation can be anywhere on the screen. But we tend to see most of the primary actions or navigation areas living at the bottom of the screen and secondary actions living at the top of the screen, with the area in between serving as the content area, like what is shown in Figure 8-8.

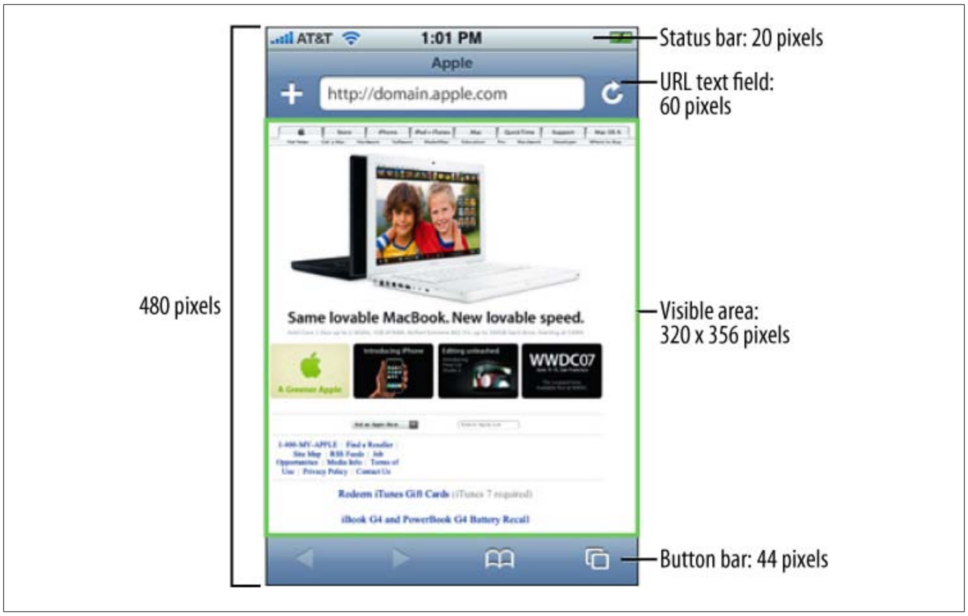


Figure 8-8. iPhone HIG, showing the layout dimensions of Safari on the iPhone

This is the opposite of the scroll navigation type, where the device's D-pad is used to go left, right, up, or down. When designing for this type of device, the primary and often the secondary actions should live at the top of the screen. This is so the user doesn't have to press down dozens of times to get to the important stuff. In Figure 8-9, you can actually see by the bold outline that the first item selected on the screen is the link around the logo.

When dealing with scroll navigation, you also have to make the choice of whether to display navigation horizontally or vertically. Visually, horizontally makes a bit more sense, but when you consider that it forces the user to awkwardly move left and right, it can quickly become a bit cumbersome for the user to deal with. There is no right or wrong way to do it, but my advice is just to try and keep it as simple as possible.

Fixed versus fluid

Another layout consideration is how your design will scale as the device orientation changes, for example if the device is rotated from portrait mode to landscape and vice versa. This is typically described as either being fixed (a set number of pixels wide), or fluid (having the ability to scale to the full width of the screen regardless of the device orientation).

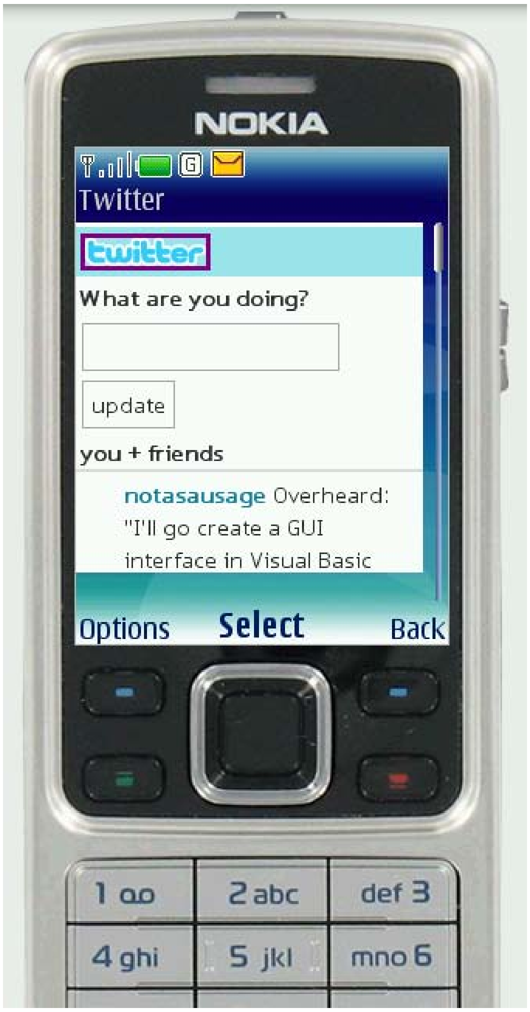


Figure 8-9. Example layout of a scroll-based application, where the user had to press the D-pad past each link to scroll the page

Orientation switching has become commonplace in mobile devices, and your design should always provide the user with a means to scale the interface to take full advantage of screen real estate.

Color

The fifth design element, color, is hard to talk about in a black-and-white book. Maybe it is fitting, because it wasn't that long ago that mobile screens were available only in black and white (well, technically, it was black on a green screen). These days, we have nearly the entire spectrum of colors to choose from for mobile designs.

The most common obstacle you encounter when dealing with color is mobile screens, which come in a number of different color or bit depths, meaning the number of bits (binary digits) used to represent the color of a single pixel in a bitmapped image. When complex designs are displayed on different mobile devices, the limited color depth on one device can cause banding, or unwanted posterization in the image.

For an example of posterization, the technical term for when the gradation of tone is replaced with regions of fewer tones, see in Figure 8-10 how dramatically the color depth can affect the quality of a photo or gradient, producing banding in several parts in the image.

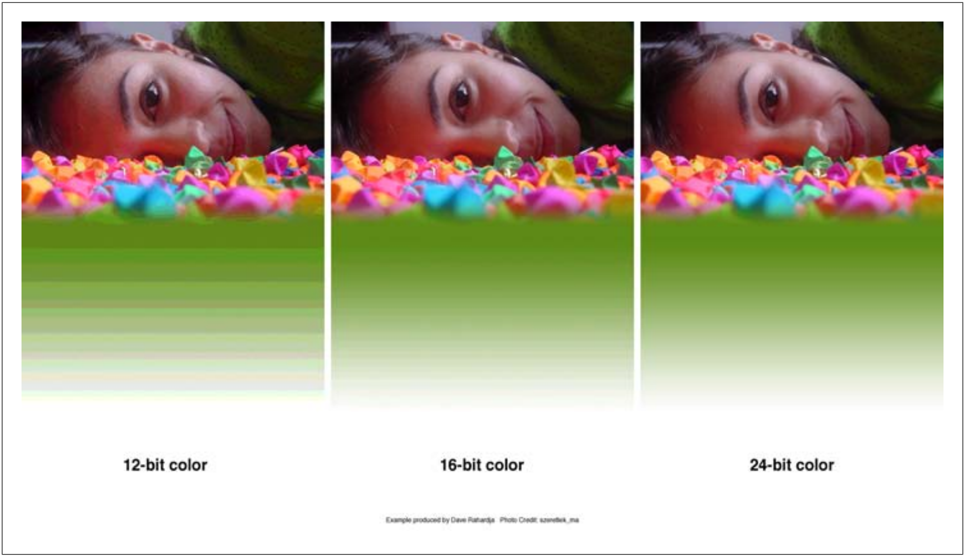

12-bit color
16-bit color
24-bit color
Example produced by Dave Rahardja Photo Credit: szeretlek_ma

Figure 8-10. An example of different levels of posterization that can occur across multiple device color depths

Different devices have different color depths. In Table 8-1, you can see the supported colors and a few example devices.

Table 8-1. Supported colors and example devices

Bit depth	Supported colors	Description	Example devices
12-bit	4,096 colors	Used with older phones; dithering artifacts in photos can easily be seen.	Nokia 6800
16-bit	65,536 colors	Also known as HighColor; very common in today's mobile devices. Can cause some banding and dithering artifacts in some designs.	HTC G1, BlackBerry Bold 9000, Nokia 6620

Bit depth	Supported colors	Description	Example devices
18-bit	262,144 colors	Used in mobile devices to offer Truecolor (see following entry) levels through dithering. Limited banding may be seen.	Samsung Alias, Sony Ericsson TM506
24-bit	16.7 million colors	Also known as Truecolor; supports millions of colors and produces little banding.	iPhone, Palm Prē, Nokia N97

The psychology of color

People respond to different colors differently. It is fairly well known that different colors produce different emotions in people, but surprisingly few talk about it outside of art school. Thinking about the emotions that colors evoke in people is an important aspect of mobile design, which is such a personal medium that tends to be used in personal ways. Using the right colors can be useful for delivering the right message and setting expectations.

One of the examples I used earlier was the ESPN mobile site, which uses a bold red header to create a stark and prominent tone to the design. But what does that say about ESPN? What does it tell the user about the experience?

For the purposes of reference, Table 8-2 provides some of the characteristics of various colors that naturally evoke certain emotions in people.

Table 8-2. Color characteristics

Color	Represents
White	Light, reverence, purity, truth, snow, peace, innocence, cleanliness, simplicity, security, humility, sterility, winter, coldness, surrender, fearfulness, lack of imagination, air, death (in Eastern cultures), life, marriage (in Western cultures), hope, bland
Black	Absence, modernity, power, sophistication, formality, elegance, wealth, mystery, style, evil, death (in Western cultures), fear, seriousness, conventionality, rebellion, anarchism, unity, sorrow, professionalism
Gray	Elegance, humility, respect, reverence, stability, subtlety, wisdom, old age, pessimism, boredom, decay, decrepitude, dullness, pollution, urban sprawl, strong emotions, balance, neutrality, mourning, formality
Yellow	Sunlight, joy, happiness, earth, optimism, intelligence, idealism, wealth (gold), summer, hope, air, liberalism, cowardice, illness (quarantine), fear, hazards, dishonesty, avarice, weakness, greed, decay or aging, femininity, gladness, sociability, friendship
Green	Intelligence, nature, spring, fertility, youth, environment, wealth, money (U.S.), good luck, vigor, generosity, go, grass, aggression, coldness, jealousy, disgrace (China), illness, greed, drug culture, corruption (North Africa), life eternal, air, earth (classical element), sincerity, renewal, natural abundance, growth
Blue	Seas, men, productiveness, interiors, skies, peace, unity, harmony, tranquility, calmness, trust, coolness, confidence, conservatism, water, ice, loyalty, dependability, cleanliness, technology, winter, depression, coldness, idealism, air, wisdom, royalty, nobility, Earth (planet), strength, steadfastness, light, friendliness, peace, truthfulness, love, liberalism (U.S. politics), and conservatism (UK, Canadian, and European politics)
Violet	Nobility, envy, sensuality, spirituality, creativity, wealth, royalty, ceremony, mystery, wisdom, enlightenment, arrogance, flamboyance, gaudiness, mourning, exaggeration, profanity, bisexuality, confusion, pride

Color	Represents
Red	Passion, strength, energy, fire, sex, love, romance, excitement, speed, heat, arrogance, ambition, leadership, masculinity, power, danger, gaudiness, blood, war, anger, revolution, radicalism, aggression, respect, martyrs, conservatism (U.S. politics), Liberalism (Canadian politics), wealth (China), and marriage (India)
Orange	Energy, enthusiasm, balance, happiness, heat, fire, flamboyance, playfulness, aggression, arrogance, gaudiness, overemotion, warning, danger, autumn, desire
Pink	Spring, gratitude, appreciation, admiration, sympathy, socialism, femininity, health, love, romance, marriage, joy, flirtatiousness, innocence and child-like qualities
Brown	Calm, boldness, depth, nature, richness, rustic things, stability, tradition, anachronism, boorishness, dirt, dullness, heaviness, poverty, roughness, earth

Note what some of the different colors can mean in different cultures. In some cases, the color you use can have opposing meanings in different cultures. This is something to consider when thinking of deploying your mobile experience to countries with the highest number of mobile devices, such as China or India.

Color palettes

Defining color palettes can be useful for maintaining a consistent use of color in your mobile design. Color palettes typically consist of a predefined number of colors to use throughout the design. Selecting what colors to use varies from designer to designer, each having different techniques and strategies for deciding on the colors. I've found that I use three basic ways to define a color palette:

Sequential
: In this case, there are primary, secondary, and tertiary colors. Often the primary color is reserved as the "brand" color or the color that most closely resembles the brand's meaning. The secondary and tertiary colors are often complementary colors that I select using a color wheel.

Adaptive
: An adaptive palette is one in which you leverage the most common colors present in a supporting graphic or image. When creating a design that is meant to look native on the device, I use an adaptive palette to make sure that my colors are consistent with the target mobile platform.

Inspired
: This is a design that is created from the great pieces of design you might see online, as shown in Figure 8-11, or offline, in which a picture of the design might inspire you. This could be anything from an old poster in an alley, a business card, or some packaging. When I sit down with a new design, I thumb through some of materials to create an inspired palette. Like with the adaptive palette, you actually extract the colors from the source image, though you should never ever use the source material in a design.

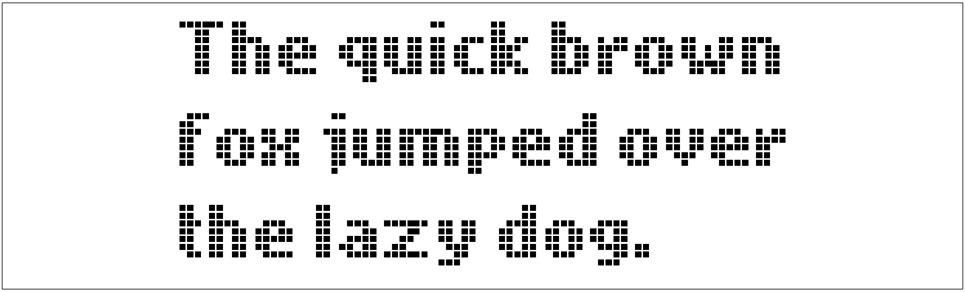


Figure 8-12. What most mobile designers think of when it comes to mobile typography

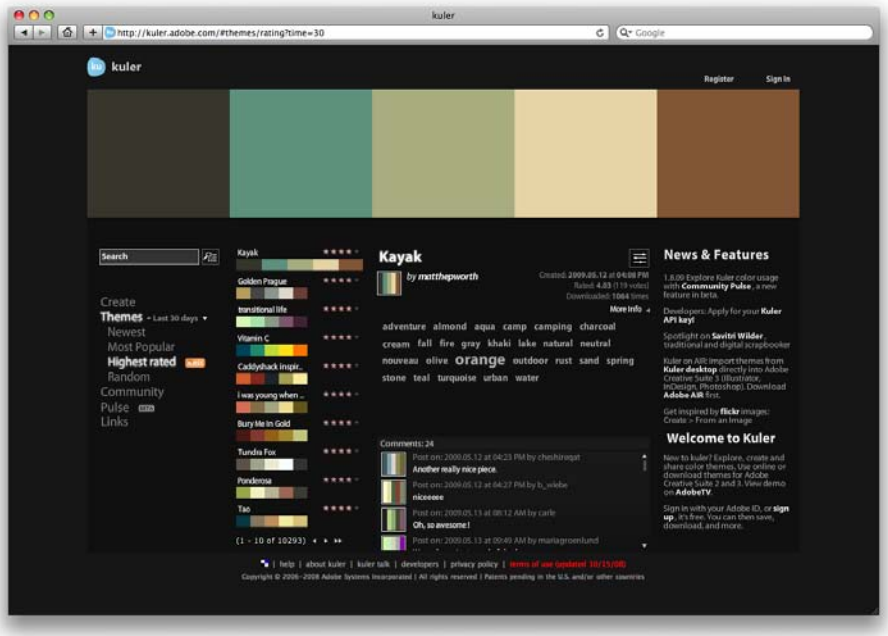


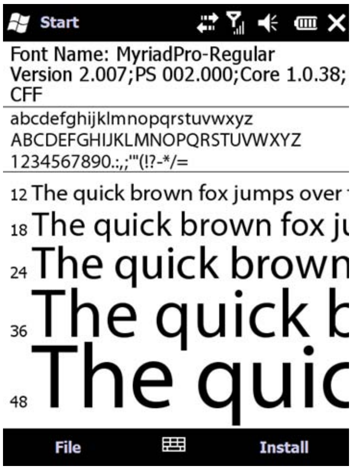
Figure 8-11. Adobe Kuler, a site that enables designers to share and use different color palettes

Typography

The sixth element of mobile design is typography, which in the past would bring to mind the famous statement by Henry Ford:

> Any customer can have a car painted any color that he wants so long as it is black.

Traditionally in mobile design, you had only one typeface that you could use (Figure 8-12), and that was the device font. The only control over the presentation was the size.



Font Name: MyriadPro-Regular
Version 2.007;PS 002.000;Core 1.0.38;
CFF

abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
1234567890.:,;'"(!?-*/=

12 The quick brown fox jumps over
18 The quick brown fox ju
24 The quick brown
36 The quick b
48 The quic

Figure 8-14. Microsoft ClearType using subpixels to display sharp text

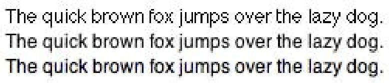
As devices improved, so did their fonts. Higher-resolution screens allowed for a more robust catalog of fonts than just the device font. First, let's understand how mobile screens work.

Subpixels and pixel density

There seem to be two basic approaches to how type is rendered on mobile screens: using subpixel-based screens or having a greater pixel density or pixels per inch (PPI).

A subpixel is the division of each pixel into a red, green, and blue (or RGB) unit at a microscopic level, enabling a greater level of antialiasing for each font character or glyph. The addition of these RGB subpixels enables the eye to see greater variations of gray, creating sharper antialiasing and crisp text.

In Figure 8-13, you can see three examples of text rendering. The first line shows a simple black and white example, the second shows text with grayscale antialiasing, and the third line shows how text on a subpixel display would render.



The quick brown fox jumps over the lazy dog.
The quick brown fox jumps over the lazy dog.
The quick brown fox jumps over the lazy dog.

Figure 8-13. Different ways text can render on mobile screens

The Microsoft Windows Mobile platform uses the subpixel technique with its ClearType technology, as shown in Figure 8-14.

The second approach is to use a great pixel density, or pixels per inch. We often refer to screens by either their actual physical dimensions ("I have a 15.4-inch laptop screen") or their pixel dimensions, or resolution (“The resolution of my laptop is 1440×900 pixels”). The pixel density is determined by dividing the width of the display area in pixels by the width of the display area in inches. So the pixel density for my 15.4-inch laptop would be 110 PPI. In comparison, a 1080p HD television has a PPI of 52.

As this applies to mobile devices, the higher the density of pixels, the sharper the screen appears to the naked eye. This guideline especially applies to type, meaning that as text is antialiased on a screen with a high density of tiny pixels, the glyph appears sharper to the eye. Some mobile screens have both a high PPI and subpixel technology, though these are unnecessary together.

Table 8-3 provides the dimensions and PPI for a few mobile devices.

Table 8-3. Dimensions and PPI for some mobile devices

Mobile device	Diagonal	Pixels	PPI
Nokia N95	2.6"	240×320	153
Apple iPhone 3G	3.5"	320×480	163
Amazon Kindle	6.0"	600×800	167
HTC Dream	3.2"	320×480	181
Sony Ericsson W880i	1.8"	240×320	222
Nokia N80	2.1"	352×416	256

Type options

Fortunately, today’s mobile devices have a few more options than a single typeface, but the options are still fairly limited. Coming from web design, where we have a dozen or so type options, the limited choices available in mobile design won’t come as a big surprise. Essentially, you have a few variations of serif, sans-serif, and monospace fonts, and depending on the platform, maybe a few custom fonts (Figure 8-15).

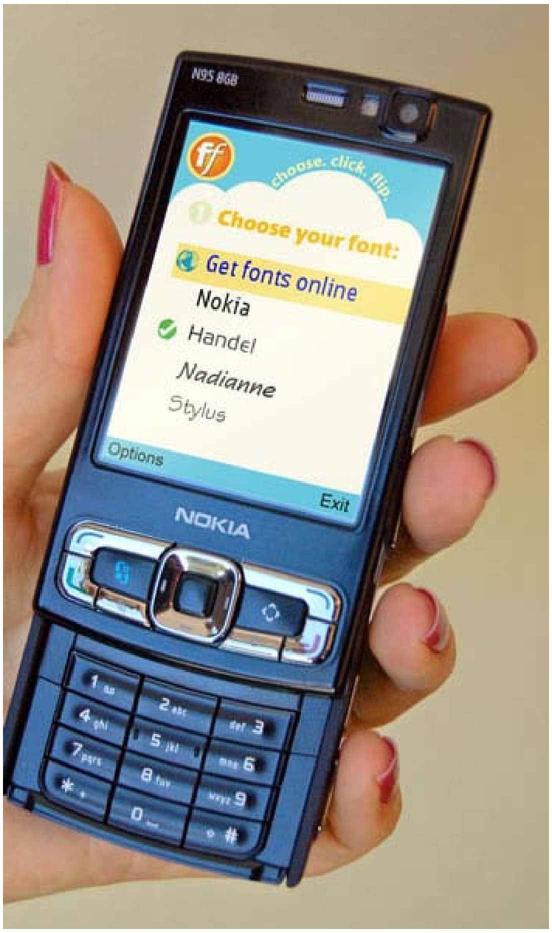


Figure 8-15. Options in typography increase as the devices become more sophisticated

In researching this book, I scoured the Web and tapped my mobile community resources to find a list of the typefaces that are included in each of the major device platforms, but I could only come up with a few—nothing close to a complete list. This goes to show how far behind mobile typography is, that designers don't even have a basic list to work from.

Therefore, when creating mobile designs for either web or native experiences, my advice is to stick with either the default device font, or web-safe fonts—your basic serif variants like Times New Roman and Georgia or sans-serif typefaces like Helvetica, Arial, or Verdana.

Font replacement

The ability to use typefaces that are not already loaded on the device varies from model to model and your chosen platform. Some device APIs will allow you to load a typeface into your native application. Some mobile web browsers support various forms of font replacement; the two most common are sIFR and Cufon. sIFR uses Flash to replace HTML text with a Flash representation of the text, but the device of course has to support Flash. Cufon uses JavaScript and the canvas element draws the glyphs in the browser, but the device of course needs to support both JavaScript and the canvas element.

In addition, the `@font-face` CSS rule allows for a typeface file to be referenced and loaded into the browser, but a license for web use is usually not granted by type foundries.

Readability

The most important role of typography in mobile design is to provide the user with excellent readability, or the ability to clearly follow lines of text with the eye and not lose one's place or become disoriented, as shown in Figure 8-16. This can be done by following these six simple rules:

Use a high-contrast typeface
: Remember that mobile devices are usually used outside. Having a high-contrast typeface with regard to the background will increase visibility and readability.

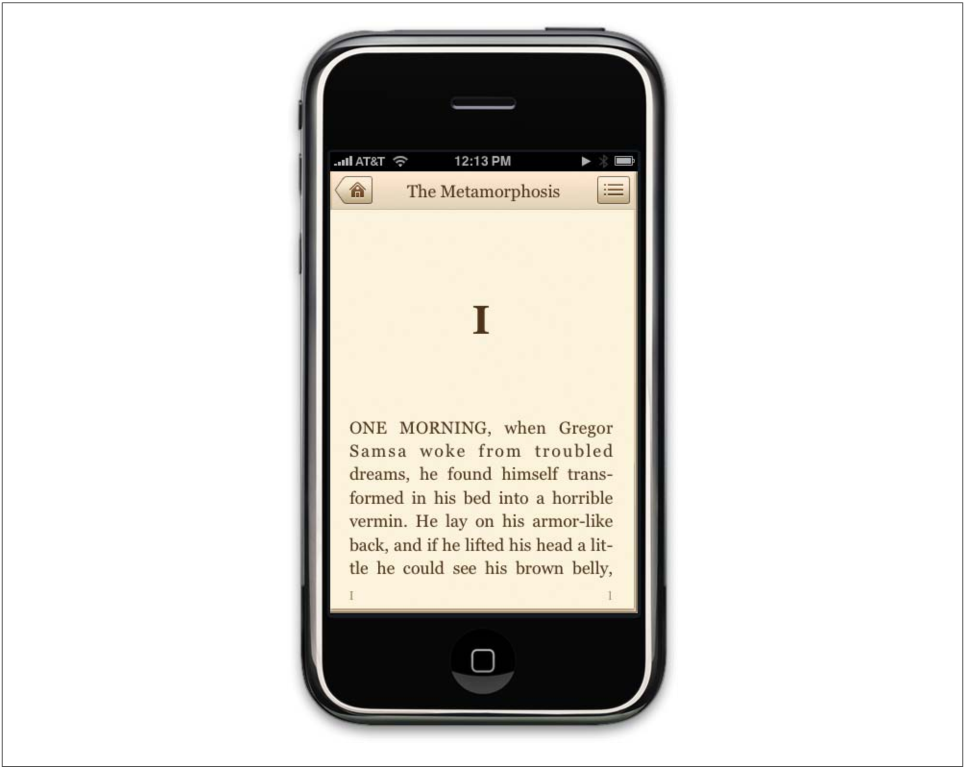
Use the right typeface
: The type of typeface you use tells the user what to expect. For example, a sans-serif font is common in navigation or compact areas, whereas serif typefaces come in handy for lengthy or dense content areas.

Provide decent leading (rhymes with "heading") or line spacing
: Mobile screens are often held 10–12" away from the eye, which can make tracking each line difficult. Increase the leading to avoid having the users lose their place.

Leave space on the right and left of each line; don't crowd the screen
: Most mobile frameworks give you full access to the screen, meaning that you normally need to provide some spacing between the right and left side of the screen's edge and your text—not much, typically about three to four character widths.

Generously utilize headings
: Break the content up in the screen, using text-based headings to indicate to the user what is to come. Using different typefaces, color, and emphasis in headings can also help create a readable page.

Use short paragraphs
: Like on the Web, keep paragraphs short, using no more than two to three sentences per paragraph.


.ııll AT&T 12:13 PM

The Metamorphosis

I

ONE MORNING, when Gregor Samsa woke from troubled dreams, he found himself transformed in his bed into a horrible vermin. He lay on his armor-like back, and if he lifted his head a little he could see his brown belly,

I 1

Figure 8-16. Classics, an iPhone application designed with readability and typography in mind

Graphics

The final design element is graphics, or the images that are used to establish or aid a visual experience. Graphics can be used to supplement the look and feel, or as content displayed inline with the text.

For example, in Figure 8-17, you can see Ribot's Little Spender application for the iPhone and the S60 platform. The use of graphical icons in the iPhone experience helps to establish a visual language for the user to interact with to quickly categorize entries. On the S60 application, the wallet photo in the upper-right corner helps communicate the message of the application to the user.

Iconography

The most common form of graphics used in mobile design is icons. Iconography is useful to communicate ideas and actions to users in a constrained visual space. The challenge is making sure that the meaning of the icon is clear to the user. For example, looking at Figure 8-18, you can see some helpful icons that clearly communicate an idea and some perplexing icons that leave you scratching your head.

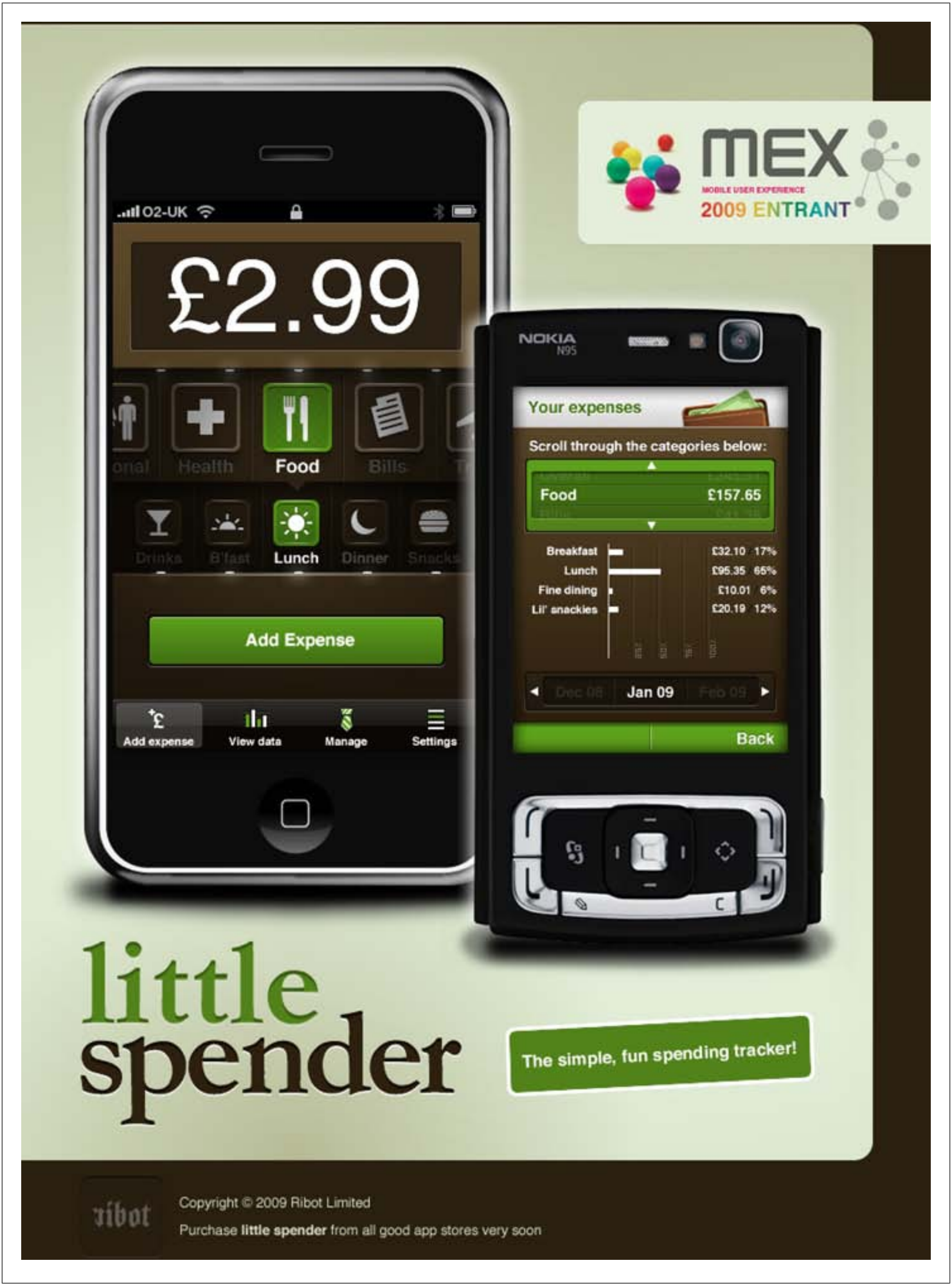
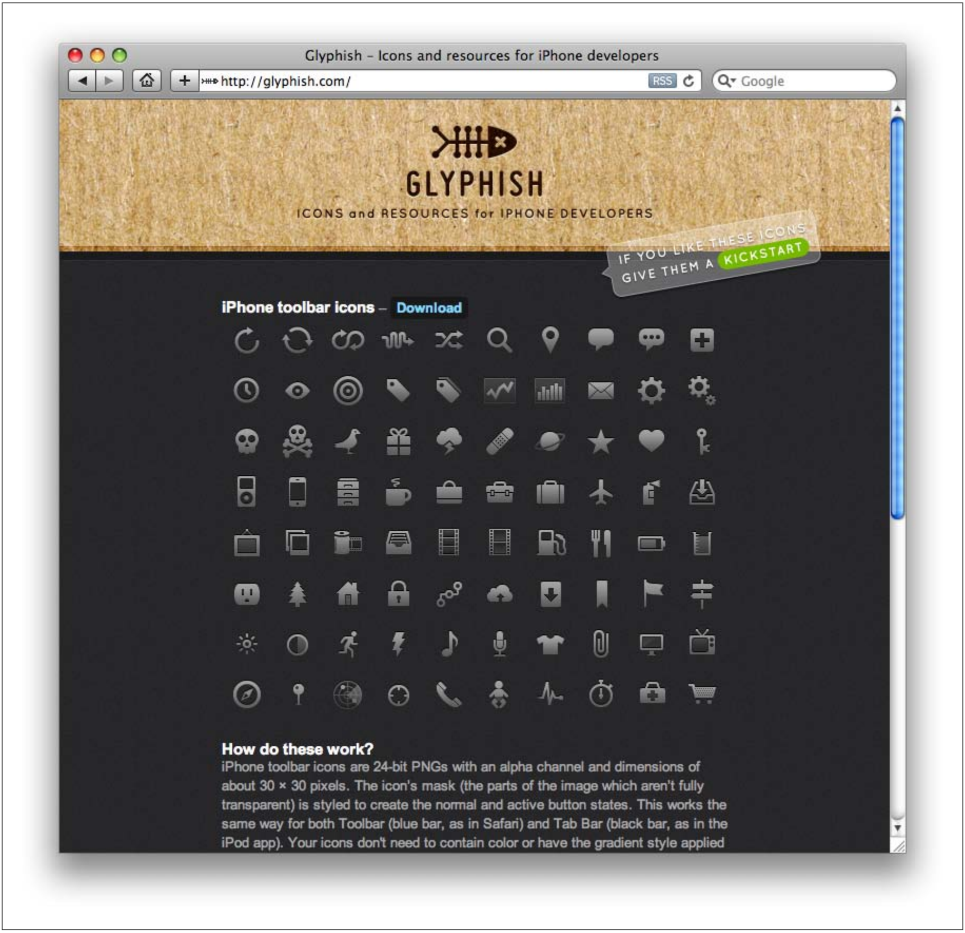


Figure 8-17. Ribot's Little Spender application uses graphics to define the experience


Glyphish – Icons and resources for iPhone developers
http://glyphish.com/
RSS
Google
GLYPHISH
ICONS and RESOURCES for IPHONE DEVELOPERS
IF YOU LIKE THESE ICONS GIVE THEM A KICKSTART
iPhone toolbar icons – Download

How do these work?
iPhone toolbar icons are 24-bit PNGs with an alpha channel and dimensions of about 30 × 30 pixels. The icon's mask (the parts of the image which aren't fully transparent) is styled to create the normal and active button states. This works the same way for both Toolbar (blue bar, as in Safari) and Tab Bar (black bar, as in the iPod app). Your icons don't need to contain color or have the gradient style applied

Figure 8-18. Glyphish provides free iPhone icons

Photos and images

Photos and images are used to add meaning to content, often by showing a visual display of a concept, or to add meaning to a design. Using photos and images isn't as common in mobile design as you might think. Because images have a defined height and width, they need to be scaled to the appropriate device size, either by the server, using a content adaptation model, or using the resizing properties of the device. In the latter approach, this can have a cost in performance. Loading larger images takes longer and therefore costs the user more.

Using graphics to add meaning to a design can be a useful visual, but you can encounter issues regarding how that image will display in a flexible UI—for example, when the device orientation is changed. In Figure 8-19, you can see how the pig graphic is designed to be positioned to the right regardless of the device orientation.

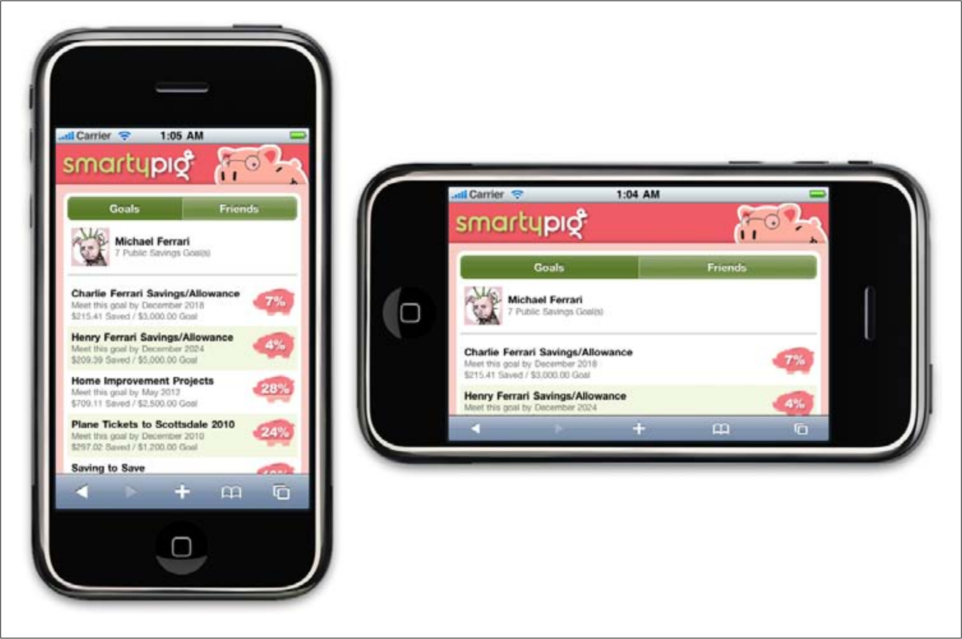

Carrier
1:05 AM
smartypig
Goals
Friends
Michael Ferrari
Charlie Ferrari Savings/Allowance
7%
Henry Ferrari Savings/Allowance
4%
Home Improvement Projects
28%
Plane Tickets to Scottsdale 2010
24%
Saving to Save
1:04 AM

Figure 8-19. Using graphics in multiple device orientations

Mobile Design Tools

As I mentioned earlier, mobile design requires understanding the design elements and specific tools. The closest thing to a common design tool is Adobe Photoshop, though each framework has a different method of implementing the design into the application. Some frameworks provide a complete interface toolkit, allowing designers or developers to simply piece together the interface, while others leave it to the designer to define from scratch.

In Table 8-4, you can see each of the design tools and what interface toolkits are available for it.

Table 8-4. Design tools and interface toolkits

Mobile framework	Design tool	Interface toolkits
Java ME	Photoshop, NetBeans	JavaFX, Capuchin
BREW	Photoshop, Flash	BREW UI Toolkit, uiOne, Flash
Flash Lite	Flash	Flash Lite
iPhone	Photoshop, Interface Builder	iPhone SDK

Mobile framework	Design tool	Interface toolkits
Android	Photoshop, XML-based themes	Android SDK
Palm webOS	Photoshop, HTML, CSS, and JavaScript	Mojo SDK
Mobile web	Photoshop, HTML, CSS, and JavaScript	W3C Mobile Web Best Practices
Mobile widgets	Photoshop, HTML, CSS, and JavaScript	Opera Widget SDK, Nokia Web Runtime
Mobile web apps	Photoshop, HTML, CSS, and JavaScript	iUI, jQTouch, W3C Mobile Web App Best Practices

Designing for the Right Device

With the best possible experience at hand, take a moment to relish it. Remind yourself that you are working with a rapidly evolving medium and though it might not be possible for every user to experience things exactly the way you've intended, you've set the tone and the vision for how the application should look. The truly skilled designer doesn't create just one product—she translates ideas into experiences. The spirit of your design should be able to be adapted to multiple devices.

Now is the time to ask, "What device suits this design best? What market niche would appreciate it most? What devices are the most popular within that niche?" The days of tent-poles are gone. Focus instead on getting your best possible experience to the market that will appreciate it most. It might not be the largest or best long-term market, but what you will learn from the best possible scenario will tell you volumes about your mobile product's potential for success or failure. You will learn which devices you need to design for, what users really want, and how well your design works in the mobile context.

This knowledge will help you develop your porting and/or adaptation strategy, the most expensive and riskiest part of the mobile equation. For example, if you know that 30 percent of your users have iPhones, then that is a market you can exploit to your advantage. iPhone users consume more mobile content and products than the average mobile user. This platform has an easy-to-learn framework and excellent documentation, for both web and native products, and an excellent display and performance means. Although iPhone users might not be the majority of your market, the ability to create the best possible design and get it in front of those users presents the least expensive product to produce with the lowest risk.

With a successful single device launch, you can start to adapt designs from the best possible experience to the second best possible experience, then the third, and fourth, and so on. The best possible experience is how it should be, so it serves as a reference point for how we will adapt the experience to suit more devices.

Designing for Different Screen Sizes

Mobile devices come in all shapes and sizes. Choice is great for consumers, but bad for design. It can be incredibly difficult to create that best possible experience for a plethora of different screen sizes. For example, your typical feature phone might only be 140 pixels wide, whereas your higher-end smartphone might be three to four times wider.

Landscape or portrait? Fixed width or fluid? Do you use one column or two? These are common questions that come up when thinking about your design on multiple screen sizes. The bad news is that there is no simple answer. How you design each screen of content depends on the scope of devices you look to support, your content, and what type of experience you are looking to provide. The good news is that the vast majority of mobile device screens share the same vertical or portrait orientation, even though they vary greatly in dimension, as shown in Figure 8-20.

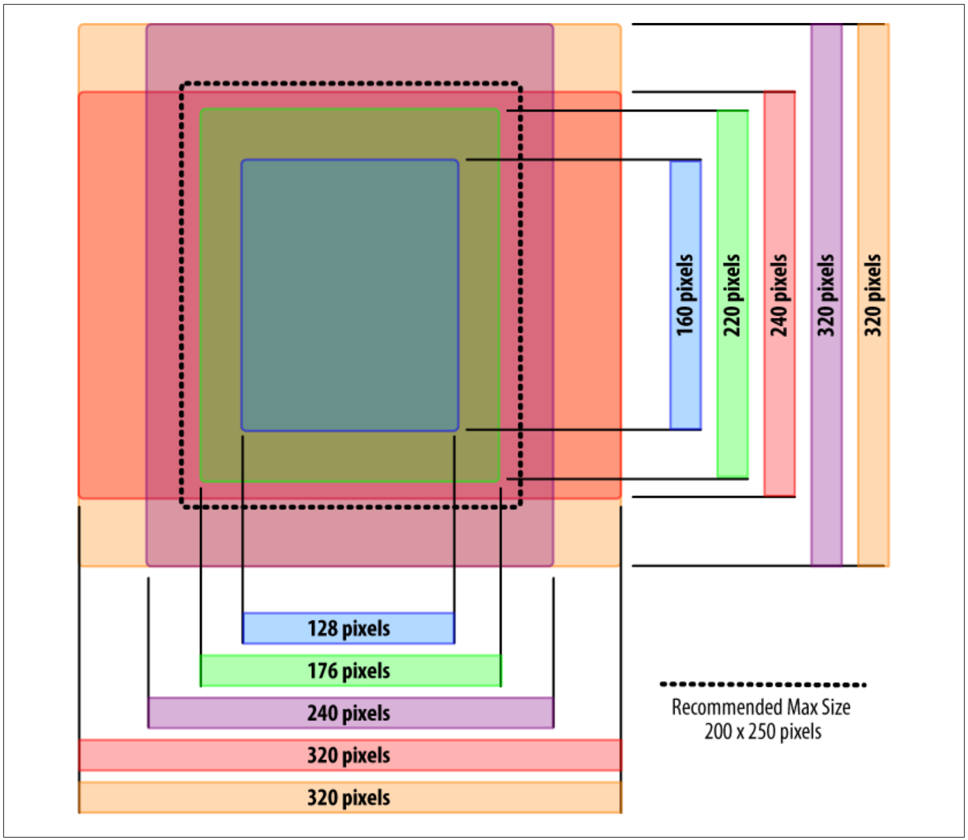


Figure 8-20. Comparing the various screen sizes

Of course, there are some devices by default in a horizontal orientation, and many smartphones that can switch between the two orientations, but most people use their mobile devices in portrait mode. This is a big shift in thinking if you are coming from interactive design, as up to this point, screens have been getting wider, not taller.

For years now, we've become used to placing less-crucial information along the sides of web pages. In software, tasks flow from left to right. With vertical designs, the goal is to think of your design as a cascade of content from top to bottom (Figure 8-21), similar to a newspaper. The most contextual information lives at the top, and the content consumes the majority of the screen. Any exit points live at the bottom. Mobile is no different.

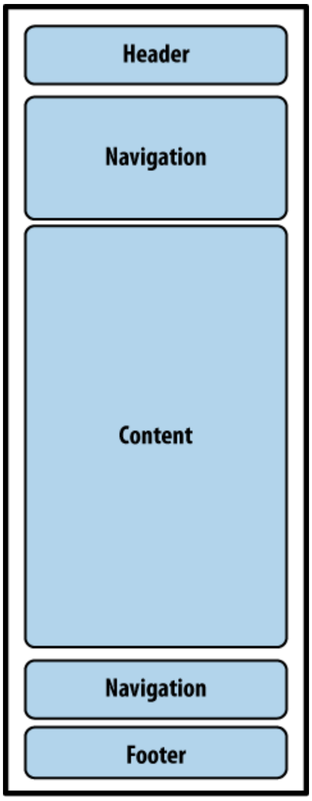


Figure 8-21. The typical flow of information on mobile devices

The greatest challenge to creating a design that works well on multiple screen sizes is filling the width. For content-heavy sites and applications, the width of mobile devices is almost the perfect readability, presenting not too many words per line of text. The problem is when you have to present a number of tasks or actions. The easiest and most compatible way is to present a stacked list of links or buttons, basically one action per line. It isn't the most effective use of space, but presenting too many actions on the horizontal axis quickly clutters the design—not to mention that it is more difficult to adapt to other devices.

Unfortunately, it isn't always reasonable to implement fluid or flexible designs that stretch to fit the width of the screen. Although most mobile web browsers and device framework APIs enable it in principle, its execution across multiple devices is a little anticlimatic. Mobile websites usually employ a fixed-width layout for the lowest common denominator, and native applications are often resized for multiple screen sizes during development.

As devices get larger, denser screens, you will see an increase in the use of touch, forcing the size of content to increase to fingertip size—typically 40 pixels wide and 40 pixels tall (Figure 8-22). This actually solves part of the horizontal axis problem, simply by making content larger for larger screens. Ironically, you can fit almost the same amount of usable content in an iPhone as you can a lower-end device.

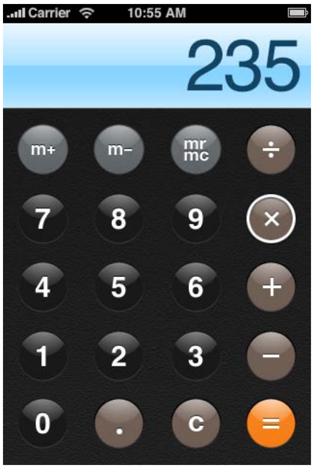


Figure 8-22. The iPhone calculator application uses common fingertip-size controls

Obviously, you can fit a lot more on screen with more advanced devices, but you want to avoid forcing the user to zoom in and out of your interfaces.

CHAPTER 10

Mobile 2.0

You've probably heard the term "Web 2.0." Although it's a commonly used term, most people you ask in the web business can't tell you what it means. To put it simply, it is just a label for the second generation of the web industry. But more importantly, it is meant to denote a change in that what we now believe in and stand for, which is not as it was before. The suffix 2.0 in technology implies that a product is new and improved, reinvented to be better and maybe even more relevant.

In the 1990s, we saw the adoption of the first generation of web technology, allowing businesses to create websites focused on the consumer market at that time. The "dot-com boom" was not about the Web; it was actually more of a boom in networking computers via the Internet, mostly driven by desire to use email as a communication and productivity tool.

In the early 2000s, the Web found its own voice. Though the technology had only incrementally evolved, the production costs dropped at the same time the market increased, creating exciting new opportunities to increase communications and productivity. Personal publishing became simple and easy to do, allowing more people to share more information and new ideas built on common problems to gain wider traction and adoption. The result was a fundamental change in how we create products for the Web, including everything from how we code, to how we design, even down to how we do business.

A few years ago, the mobile community started to discuss the idea of "Mobile 2.0," borrowing from many of the same principles behind Web 2.0. Each of these principles serves to transform the Web into a more agile and user-centered medium for delivering information to the masses. Mobile development, under the bottlenecks of device fragmentation and operator control, is sorely in need of a little reinvention as well.

Following is a recap of the original seven principles of Web 2.0:

The Web as a platform
: For the mobile context, this means "write once, deploy everywhere," moving away from the costly native applications deployed over multiple frameworks and networks.

Harnessing collective intelligence
: This isn't something the mobile community has done much of, but projects like WURFL—an open source repository of device profiles provided by the community—is exactly what mobile needs more of.

Data is the next Intel inside
: Mobile takes this principle several steps further. It can include the data we seek, the data we create, and the data about or around our physical locations.

End of the software release cycle
: Long development and testing cycles heavily weigh on mobile projects, decreasing all hopes of profitability. Shorter agile cycles are needed to make mobile development work as a business. Releasing for one device, iterating, improving, and then releasing for another is a great way to ensure profitability in mobile.

Lightweight programming models
: Because mobile technology is practically built on enterprise Java, the notion of using lightweight models is often viewed with some skepticism. But decreasing the programming overhead required means more innovation occurs faster.

Software above the level of a single device
: This effectively means that software isn't just about computers anymore. We need to approach new software as though the user will demand it work in multiple contexts, from mobile phones to portable gaming consoles and e-book readers.

Rich user experiences
: A great and rich user experience helps people spend less time with the software and more time living their lives. Mobile design is about enabling users to live their lives better.

Although the mobile industry has been through many more evolutions than just two, the concepts behind Web 2.0 are some of the most important ideas in not just mobile technology, but the Web as a whole.

What Is Mobile 2.0?

In autumn 2006, I was asked to help design and build a website for the first Mobile 2.0 conference, happening a few days before O'Reilly's Web 2.0 Summit. The event was put together by several Mobile Monday organizers. Mobile Monday is a series of mobile social gatherings that happen all over the world on the first Monday of the month. The idea was to bring some of the greatest minds in the mobile field together in San Francisco to attend the Web 2.0 Summit in order to discuss the future of mobile development.

By the time the event arrived, I had become frustrated with the term Web 2.0. By this time, it had devolved from a great idea to jargon that people would casually toss about. Everyone wanted "Web 2.0," but no one understood what it meant. Needless to say, I was skeptical about anything labeled 2.0. So I went to the first Mobile 2.0, curious about exactly what my fellow speakers thought Mobile 2.0 actually meant.

By the end of the day, I had a loosely defined picture of what Mobile 2.0 was, and for the first time in a long time I was excited about the future of mobile development. I saw a groundswell from the people at the conferences to finally overcome the industry bureaucracy and technical problems that had gone unsolved for too long. I saw not only what mobile technology could be, but people willing to make it happen.

In an interesting twist of foreshadowing, the first Mobile 2.0 event occurred only a few months before the announcement of the first iPhone—a device that would come to symbolize many of the things we talked about that day.

I took away many principles about Mobile 2.0, and about the future of mobile, which still hold true today. Like in Web 2.0, these ideas cover the industry from principles to techniques. The principles of Mobile 2.0 discussed in the following sections try to define the direction mobile development is headed in, sometimes by calling out the problems the medium faces still to this day.

Mobile 2.0: The Convergence of the Web and Mobile

It is obvious that in the minds of many, Mobile 2.0 is the Web. At this point, the mobile web has always been viewed as a second-class citizen within the mobile ecosystem, for many reasons, as discussed later.

Mobile is already a medium, but the consensus is that by leveraging the power of the Web, integrating web services into the mobile medium is the future of mobile development. When the iPhone exploded onto the scene, it increased the usage of the mobile web by its users to levels never seen before. The spur of new mobile web apps created just for the iPhone doubled the number of mobile websites available in under a year.

If Web 2.0 taught us that the Web is the platform, then Mobile 2.0 tells us that mobile will be the primary context in which we leverage the Web in the future.

The Mobile Web Browser As the Next Killer App

If the future of mobile is the Web, then it only makes sense that the mobile web browser is the next killer app of mobile. Again, this is something we saw confirmed with WebKit in the iPhone and later in Android.

However, of particular concern is how device fragmentation factors into mobile browsers. For example, how can we expect developers to support more than 30 different mobile browsers? A fellow panelist from the Mozilla Minimo project offered a potential solution in consolidation—that we will see only a few mobile browsers in the future; specifically, browsers built on Mozilla, Opera, Internet Explorer, and WebKit technologies. At the time, I thought that prediction was too focused on smartphones, but in the years since, the line between smartphone and feature phone seems to be going away, so this prediction is fairly accurate.

But the single biggest challenge in mobile remains device fragmentation. The mobile browser enables us to penetrate the problem by not having our content locked so specifically to the device abilities, screen size, and form factor, but device fragmentation still causes old, outdated browsers to remain in the market long after they should be put out to pasture.

What appears to be solving browser fragmentation is actually the iPhone. The Mobile Safari browser included with the iPhone provided such an excellent web experience on a mobile device that it drove use of the mobile web to huge levels, which means big profits for the operators. This also means that the mobile web is no longer a second-class citizen. In the post-iPhone market, all new devices are judged by the quality of their mobile web browser. Operators know it and therefore are demanding better browsers from device makers and browser makers.

Mobile Web Applications Are the Future

Creating mobile web applications instead of mobile software applications has remained an area of significant motivation and interest. The mobile community is looking at the Web 2.0 revolution for inspiration, being able to create products and get them to market quickly and at little cost. They see the success of small iterative development cycles and want to apply this to mobile development, something that is not that feasible in the traditional mobile ecosystem.

Developers have been keen for years to shift away from the costly mobile applications that are difficult to publish through the mobile service provider, require massive testing cycles and costly porting to multiple devices, and can easily miss the mark with users after loads of money have been dumped into them.

The iPhone App Store and the other mobile device marketplaces have made it far easier to publish and sell, but developers still have to face difficult approval processes, dealing with operator and device maker terms and porting challenges.

Mobile software has two fundamental problems that mobile web applications solve. The first is forcing users through a single marketplace. We know from years of this model that an app sold through a marketplace can earn huge profits if promoted correctly. Being *promoted correctly* is the key phrase. What gets promoted and why is a nebulous process with no guarantees. One thing is certainly true: the companies that know how to work the system are the ones that get the big prizes, making it increasingly hard for the small developer to see any kind of success. But the mobile web provides any size of developer with the ability to promote and distribute their app on their terms, building a relationship directly with their customers and not by proxy.

The second problem is the ability to update your application. It is certainly possible on modern marketplaces like the App Store, but we are still years from that being the norm. Mobile web apps enable you to make sure that you never ship a broken app, or if your app breaks in the future due to a new device, to be able to fix it the same day the device hits the street. This flexibility isn't possible in the downloaded app market.

JavaScript Is the Next Frontier

If you are going to provide mobile web applications, you have to have a mobile web browser that supports Ajax, or, as it is technically known, XMLHttpRequest. It makes a lot of those cool interactions in your web browser work via the capability to load content asynchronously into your browser view.

But it isn't just Ajax; it is JavaScript, a web technology that has largely been ignored with most mobile web browsers. Ajax is great, but just being able to do a little show/hide or change a style after you click or touch it goes a long way toward improving the user experience.

This is probably where mobile web browsers fall behind desktop browsers the most. Because they both support XHTML and CSS relatively well, JavaScript has been a no-go in mobile for years. In order for mobile web apps to rival native applications, you have to support some JavaScript.

Modern mobile browsers have made much progress over the last few years, but there is still plenty of work to be done. For example, accessing the device capabilities like the phone book or filesystem with JavaScript doesn't work in a consistent way. These problems still need to be solved in order to truly reap the benefits of the Web.

Rich interactions kill battery life

JavaScript and Ajax have been ignored because using an Ajax-based web application on your phone can drain your battery at a rate of four to five times your normal power consumption. I've heard a number of reasons for why this happens from mobile hardware guys much smarter than myself, but to summarize, the two most prevalent are:

- JavaScript consumes more processor power and therefore more battery life.
- Ajax apps fetch more data from the network, meaning more use of the radio and more battery life.

Unless you are in the habit of carrying around a bunch of extra batteries, expect to charge your phone every hour or two as a penalty for using the modern mobile web.

Apple and the open source WebKit browser have made huge strides by releasing a JavaScript engine that is incredibly efficient on mobile devices, though the other big mobile browser technologies aren't far behind. This problem is going away quickly as the mobile browsers get better, batteries improve efficiency, and devices get more powerful.

The Mobile User Experience Is Awful

Traditionally, the user experience available in the mobile web has been like using a website from 1995: mostly text-based, difficult to use, and ugly as sin. This isn't to say that the user experience of mobile applications has been much better, but it used to be that if you wanted a good experience, you built a native app.

Descriptions within the industry range from the honest "the mobile user experience is utterly horrid," to the sales pitch of "look at these cool things you can do," to the optimistic "the mobile user experience is the future!" These polar attitudes toward the mobile user experience are somewhat ironic, given that the mobile user experience was largely ignored for close to a decade. People in mobile treat the user experience like a chicken-and-egg scenario: bad input/output of the user experience prevents adoption, but designing a shiny user experience with bells and whistles will bring them in droves.

Device APIs usually force you to use their models of user experience, meaning that you have to work in the constraints of the API. This is good for creating consistent experiences for that platform, but these experiences don't translate to others. For example, would you take an iPhone app design and put it on an Android device? The user experience for these devices is similar but still remains different.

The beauty of the Web, literally, is that you can design whatever experience you want, for better or worse. You are in control of the presentation and can establish your own visual metaphors. The problem has been that traditionally complex (which often equates to good) user experiences haven't been possible on mobile devices. Modern mobile web browsers, as they come closer to their desktop counterparts, remove this distinction, giving us the same canvas on mobile devices that we have for the desktop. This means that creating mobile experiences just got a whole lot easier. It also means we can have a consistent user experience across multiple mediums.

Mobile Widgets Are the Next Big Thing

At many Mobile 2.0 events, I've heard a lot of buzz about mobile widgets, though no one can tell me how mobile widgets would define a mobile widget, or how they are different from mobile web apps. The consensus seems to be that the solution for the challenges with the mobile web is to create a series of "small webs" targeted at a specific user or task. Though I couldn't figure out the problem being solved with these widgets, I had to admit that they looked pretty cool.

Don't get me wrong. I believe that the concept of small network-enabled applications is very promising, but the mobile industry tends to take promising ideas like this, inflate expectations to unsustainable levels, then abandon them at the first sign of trouble or sacrifice them for the next big thing, whichever happens first.

The mobile web is here: it works, but it just needs some love. It is the long bet. I believe we should get that sorted out before we try to add widgets on top of it.

Carrier Is the New "C" Word

I noticed a strong tendency over the years for people in the mobile industry to avoid uttering the word "carrier." Even the more European equivalent term "operator" seems to be on the decline. To give you an example of how much carriers are hated: you can have entire conversations with people who work for the big operators and even they will avoid using the term in conversation.

It is almost like when you get a bunch of mobile experts in a room for a day: they want to pretend that operators don't exist at least for one day. Maybe they prefer to see a future with no mobile service providers at all. I think more likely the case is that the industry has finally figured out that very few can make a profit when your business relies on carriers. Though the "C" word isn't uttered often, it still is the 800-pound gorilla in the room.

It is clear that one of the key drivers of Mobile 2.0 and the focus on the mobile web is to find a way to build a business that doesn't rely on carrier control.

Mobile Needs to Check Its Ego

For years, I've sat on the line between the mobile community and the web community. They have treated each other almost like rivals. I've been frustrated with both sides, but it is the mobile camp that needs to check their egos at the door and get into the game, before they learn that all the rules have changed.

On the mobile side, you have some incredibly intelligent people who have been innovating amazing products under insane constraints for years. On the web side, you have creative amateurs who have helped build a community and ecosystem out of passion and an openness to share information.

The web guys want to get into the game and move the medium forward, partly out of desire open up a new market for themselves, but mostly out of passion for all things interactive. But, to the mobile community, they are seen as a threat to expertise. On the other hand, to the web community, the mobile guys come off as overly protective, territorial, selfish, and often snobbish or egotistical, effectively saying, "Go away."

Don't get me wrong—I think that the mobile guys are very smart and great people; some of them I consider to be close friends. They have to deal with really hard problems that would make a web professional give up to go serve coffee. But I'm a very patient and tolerant person, and I have to admit that these same people are some of the most difficult people I've ever worked with.

For example, for years I wondered why I, a designer with mobile experience, have been asked to speak at so many conferences or write so many articles. I would think to myself, "I can think of 10 guys who know a lot more about mobile than I do. They should be here, not me." But then this guy who came up to me after one of my workshops said, "I've been in mobile for a long time, and have been to a lot of mobile events, but no one has ever spoken about mobile so plainly." I thanked him for the wonderful comment and replied, "Well, that's the problem with mobile, isn't it?"

My point is that unless the mobile community comes together with the web community by sharing information, experience, and guidance, one day they will find that their experience has become obsolete. In return, the web guys will share their enthusiasm, willingness to learn, and passion that many in mobile development have forgot.

It's that one principle of Web 2.0 that the mobile community has left out: harnessing collective intelligence. The Web and the mobile community are reaching a point where the two worlds can no longer afford not to be working together, sharing what they know and harnessing the collective intelligence of both media.

We Are Creators, Not Consumers

The final principle of Mobile 2.0 is recognizing that we are in a new age of consumerism. Yesterday's consumer does not look anything like today's consumer. The people of today's market don't view themselves as consumers, but rather as creators. But before we get into that, let's back up for a minute.

The web is about content. Sure, there are programming languages, APIs, and other technical underpinnings, but what do you do when you open a web browser? You read. Our primary task online is to read, to gain information. During the early days of the Web, it took tools and know-how in order to publish to the Web. But early in the Web 2.0 evolution, we saw a rise in tools that allowed us to publish to the Web easily, giving individuals a voice online, with a massive audience.

This democratization of the Web took many forms that some call "social media," like blogging, social networks, media sharing, microblogging, and lifestreams. Although social media may have many facets, they all share the same goal: to empower normal, everyday people to become creators and publishers of content. It started with the written word, then music, then photos, and more recently video was added. Entire markets have been created to provide today's consumer with gadgets, software, and web services to record and publish content so that we can share it with our friends and loved ones.

At the center of this revolution in publishing is the mobile device. As networked portable devices become more powerful, allowing us to capture, record, and share content in the moment, we are able to add a new kind of context to content—the likes of which we haven't seen since satellite television. Now you can share any moment with any group of people in real time. Think about how powerful a concept that is! It could change entire cultures.

Tony Fish, coauthor of *Mobile Web 2.0* (futuretext), says:

> When everyone has the tools to create content, in addition to zero-cost publishing, we do not consume content, we create it.[*]

In the early days of the Web, I marveled at how a networked population might change our society forever. Now I realize that the change occurs wherever the device is, the context it is within. The early "Web 1.0" days clearly changed how business is done, because businesses are the primary consumer of desktop computers. It probably is no coincidence that Web 2.0 occurred around the same time that laptop computers became affordable for the average person, making the Web a more personal medium.

With Mobile 2.0, the personal relevance of the content matches how personal the device is and how personally it applies to our everyday situations or our context. I see now that this is the time and medium that delivers on that initial promise of the Web: to change society forever.

[*] Sources: *http://communities-dominate.blogs.com/brands/2007/02/mobile_the_7th_.html* and *Mobile as 7th of the Mass Media*, by Tomi Ahonen (futuretext).

Chapter 1

In-Page Editing

Content on web pages has traditionally been display-only. If something needs editing, a separate form is presented with a series of input fields and a button to submit the change. Letting the user directly edit content on the page follows the principle of *Make It Direct*.

This chapter describes a family of design patterns[*] for directly editing content in a web page. There are six patterns that define the most common in-page editing techniques:

Single-Field Inline Edit
: Editing a single line of text.

Multi-Field Inline Edit
: Editing more complex information.

Overlay Edit
: Editing in an overlay panel.

Table Edit
: Editing items in a grid.

Group Edit
: Changing a group of items directly.

Module Configuration
: Configuring settings on a page directly.

The most direct form of **In-Page Editing** is to edit within the context of the page. First, it means we don't leave the page. Second, we do the editing directly in the page.

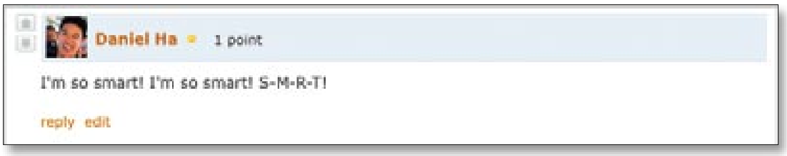
[*] We use the term "design patterns" to denote common solutions to common problems. Design patterns originate from Christopher Alexander's book *A Pattern Language* (Oxford University Press). You can read a series of essays from me (Bill) and others on design patterns at *http://www.lukew.com/ff/entry.asp?347*.

The advantage of **Inline Edit** is the power of context. It is often necessary for users to continue to see the rest of the information on the page while editing. For example, it is helpful to see the photo while editing the photo's title, as explained in the next section, "Single-Field Inline Edit."

It is also useful when editing an element that is part of a larger set. Disqus, a global comment service, provides inline editing for comments (Figure 1-1). After posting a comment and before anyone replies to the comment, an edit link is provided. The editing occurs within the context of the rest of the comments shown on the page.

Tip

If editing needs the context of the page, perform the editing inline.


Daniel Ha • 1 point

I'm so smart! I'm so smart! S-M-R-T!

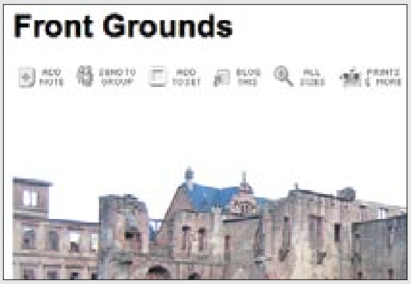
reply edit

Figure 1-1. *Disqus allows comments to editing inline within the context of other comments*

The first two patterns, **Single-Field Inline Edit** and **Multi-Field Inline Edit**, describe techniques for bringing direct inline editing into the page.

Single-Field Inline Edit

The simplest type of **In-Page Editing** is when editing a single field of text inline. The editing happens in place instead of in a separate window or on a separate page. Flickr provides us a canonical example of **Single-Field Inline Edit** (Figure 1-2).

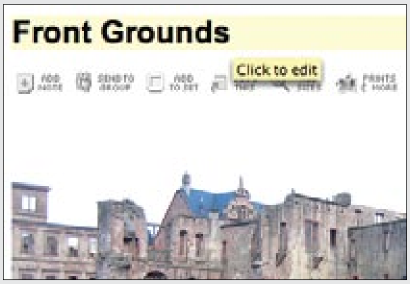


Non-editing state

Flickr chose to keep the title clear of any edit actions.

The title looks like a title. This provides the highest readability by not cluttering the interface with edit actions or an editable style.

This will lead to discoverability issues. An alternate approach is to place an "edit" link somewhere in line with the title that would start the editing process.

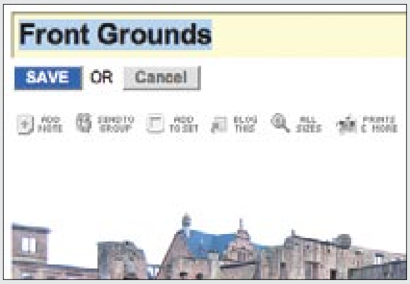


Invitation to edit

On mouse hover, the background is backlit with yellow. A tool tip invites the user to "Click to edit".

Invitations attempt to lead the user to the next level of interaction (from mouse hover to mouse click).

Invitations have to be discovered to be useful. Flickr's bet is that the user will drift his mouse over the title (of his own photo).

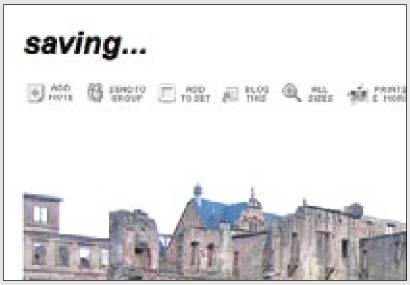


Editing

Once the user clicks on the title, it is placed into an edit mode. An edit box is switched into view immediately under the title text.

The "Save" and "Cancel" buttons make it clear we are editing the title by providing a familiar interface—the user input form.

A disadvantage to this approach is that the picture gets pushed down to make way for the additional interface elements.



Completion

There are a number of ways to signify that the text is being saved. In this example, the title text is temporarily replaced with the text "saving…". Upon completion, the new title is shown in the non-editing style.

An alternative approach is to show a busy progress indicator while the change is being made.

Figure 1-2. *Flickr provides a straightforward way to edit a photo's title directly inline*

Considerations

The flow is simple. Click on the title to start editing. When you are done, hit the "Save" button, and the title is saved in place. Flickr was one of the first sites to employ this type of in-page editing. As a testament to its usefulness, the interaction style first designed has not changed much over the last few years.

But there are some challenges to consider.

Discoverability

Just how discoverable is this feature? In this example, there are a number of cues that invite the user to edit. The invitations include:

- Showing a tool tip ("Click to edit")
- Highlighting the background of the editable area in yellow
- Changing the cursor to an edit cursor (I-beam)

But all these cues display *after* the user pauses the mouse over the title (mouse hover). Discoverability depends on the user hovering over the title and then noticing these invitations.*

To make the feature more discoverable, invitational cues could be included directly in the page. For example, an "edit" link could be shown along with the title. Clicking the link would trigger editing. By showing the link at all times, the edit feature would be made more discoverable.

But this has to be balanced with how much visual noise the page can accommodate. Each additional link or button makes the page harder to process and can lead to a feature not being utilized due to the sheer volume of features and their hints shown on the page.

Tip

If readability is more important than editability, keep the editing action hidden until the user interacts with the content.

Yahoo! Photos† took this approach for editing titles (Figure 1-3). When showing a group of photos, it would be visually noisy to display edit links beside each title. Instead, the titles are shown without any editing adornment. As the mouse hovers over a photo title, the text background highlights. Clicking on the title reveals an edit box. Clicking outside of the edit field or tabbing to another title automatically saves the change. This approach reduces the visual noise both during invitation and during editing. The result is a visually uncluttered gallery of photos.

* While the Yahoo! Design Pattern Library (*http://developer.yahoo.com/ypatterns/*) was being launched, this pattern was not included in the initial set of patterns due to an internal debate over this issue of discoverability. In fact, one of the reviewers, a senior designer and frequent user of Flickr, had only recently discovered the feature. As a result, we withheld the pattern from the public launch.

† Yahoo! Photos was replaced in 2007 with Flickr.

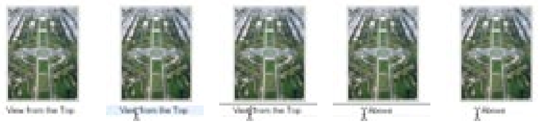


Figure 1-3. *Editing titles in Yahoo! Photos keeps visual clutter to a minimum; it simply turns on a visible edit area during editing*

Accessibility

Another concern that arises from inline editing is the lack of accessibility. Accessibility affects a wider range of users than you might first consider. Assistive technologies help those with physical impairments, medical conditions, problems with sight, and many other conditions.

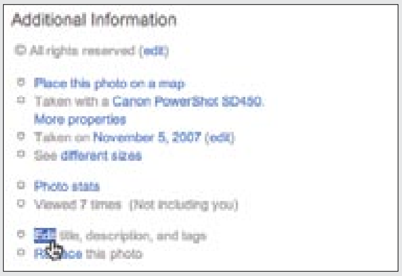
Assistive technologies generally parse the page's markup to find content, anchors, alternate titles for images, and other page structure. If the inline edit feature does not contain explicit markup built into the page (such as an explicit, visible edit link), assistive technologies cannot easily discover the inline edit feature.

In a sense, relying on the mouse to discover features will prevent some users from being able to edit inline. As mentioned before, providing an explicit edit link helps with discoverability (as shown previously in Figure 1-1). But as a by-product it also makes the feature more accessible.

Tip

Providing an alternative to inline editing by allowing editing on a separate page can improve accessibility.

There is a natural tension between direct interaction and a more indirect, easily accessible flow. It is possible to relieve this tension by providing both approaches in the same interface. Flickr actually does this by offering an alternate, separate page for editing (Figure 1-4).



Additional Information

© All rights reserved (edit)

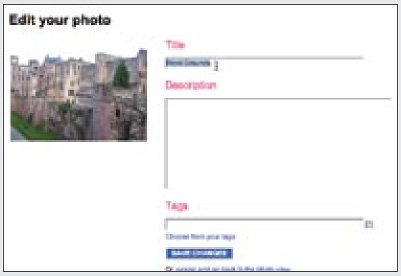
- Place this photo on a map
- Taken with a Canon PowerShot SD450.
  More properties
- Taken on November 5, 2007 (edit)
- See different sizes

- Photo stats
- Viewed 7 times (Not including you)

- Edit title, description, and tags
- Replace this photo

Edit link

Flickr provides a separate "Edit" link for editing the title, description, and tags for the photo.



Separate page for editing

The “Edit your photo” page provides a standard, form-based interface for editing. Assistive technologies work easily with static pages like this.

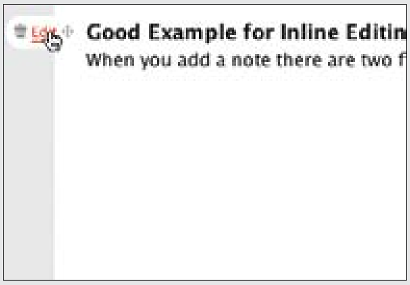
Figure 1-4. *Flickr allows you to also edit a photo’s title, description, and tags in a separate page*

Multi-Field Inline Edit

In our previous example, a single value was being edited inline. What happens if there are multiple values, or the item being edited is more complex than a string of text and you still would like to edit the values inline?

The pattern **Multi-Field Inline Edit** describes this approach: editing multiple values inline.

37 Signal’s Backpackit application uses this pattern for editing a note (Figure 1-5). A note consists of a title and its body. For readability, the title is displayed as a header and the body as normal text. During editing, the two values are shown in a form as input text fields with labeled prompts.

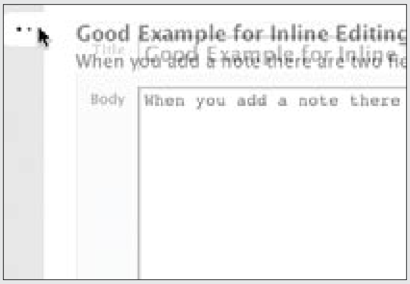


Good Example for Inline Editin

When you add a note there are two f

Invitation to edit

On mouse hover two actions are revealed: edit and delete. The “Edit” link serves as the invitation to edit the note.



Good Example for Inline Editing
When you add a note there are two fi

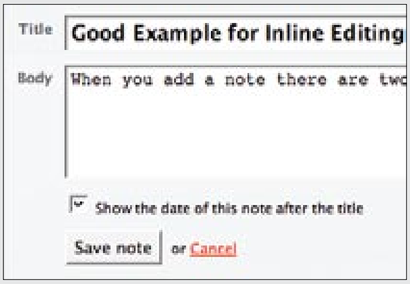
Title Good Example for Inline

Body When you add a note there

Transition to edit

Once you click the "Edit" link, a busy indicator is animated while an edit form for the title and body replaces the textual display of the note.

A slight cross-fade transition and a progress indicator are used to cue the user to this switch of context. This ties the textual display and the edit form together.



Title Good Example for Inline Editing

Body When you add a note there are twc

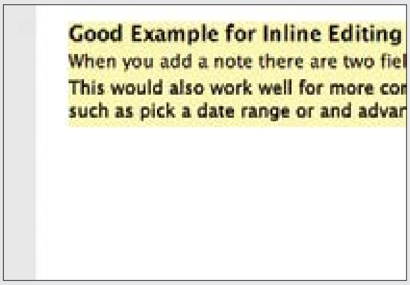
☑ Show the date of this note after the title

Save note or Cancel

Editing

Editing is very straightforward since it uses a form. It provides the flexibility for other options to be placed in the editing form.

The "Save note" button completes the edit.



Good Example for Inline Editing
When you add a note there are two fiel
This would also work well for more cor
such as pick a date range or and advar

Completion

The edit form is replaced with the new title and body. To make it clear that a change occurred, the background is highlighted in yellow. After a few seconds the yellow is faded back down to the normal background color.

Contrast this with Flickr's technique of showing status while saving instead of after completion.

Figure 1-5. *Backpackit reveals a multi-field form for editing a note's title and body*

Considerations

In **Single-Field Inline Edit** the difference between display mode and edit mode can be more easily minimized, making the transition less disruptive. But when editing multiple fields, there is a bigger difference between what is shown in display mode and what is needed to support editing.

Readability versus editability

Readability is a primary concern during display. But the best way to present editing is with the common input form. The user will need some or all of the following:

- Edit controls
- Field prompts
- Help text for user input
- Error handling
- Assistive input (e.g., calendar pop up or drop-down selection field)
- Editing styles (e.g., edit fields with 3D sunken style)

The edit mode will need to be different in size and layout, as well as in the number and type of components used. This means that moving between modes has the potential to be a disruptive experience.

In our example, the form for editing the note takes up a larger space on the page than when just displaying the note.

Blending display and edit modes

Ideally you would like the two modes to blend together in a seamless manner. Bringing the edit form into the page flow will have an effect on the rest of the page content. One way to smooth out the transition is by a subtle use of animation. Backpackit does this by fading out the display view and fading in the edit view at the same time (see the cross-fade in Figure 1-5).

Another approach is to use the same amount of space for both display and edit modes. In Yahoo! 360, you can set a status message for yourself. Your current status shows up on your 360 home page as a "blast," represented as a comic book-style word bubble. Visually it looks like a single value, but there are actually three fields to edit: the blast style, the status, and any web page link you want to provide when the user clicks on your blast. Figure 1-6 shows the blast as it appears before editing.

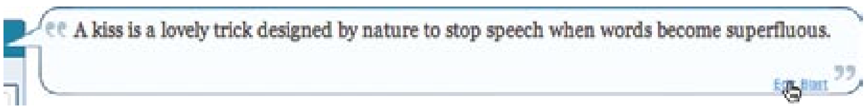
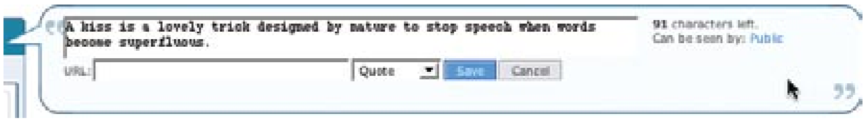

A kiss is a lovely trick designed by nature to stop speech when words become superfluous.

Figure 1-6. *Yahoo! 360 shows an "Edit Blast" link to invite editing*

Figure 1-7 shows how the blast appears during editing. Notice that the edit form is designed to show both modes (display and editing) in the same visual space.


A kiss is a lovely trick designed by nature to stop speech when words become superfluous.

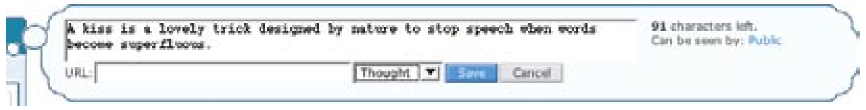
URL:

Figure 1-7. *Yahoo! 360 brings the editing into the displayed blast; the difference between display and edit modes is minimized*

The size similarity was not an accident. During design there was a concerted effort to make the display mode slightly larger without losing visual integrity, while accommodating the editing tools in the space of the bubble.

WYSIWYG*

If the two modes (display and editing) are in completely separate spaces, the user may lose a sense of what effect the change will have during display. In Yahoo! 360, you can change the type of bubble and immediately see what it will look like. Switching from a "quote" bubble to a "thought" bubble is reflected while still in editing mode (Figure 1-8). This would not be possible if editing happened in a separate edit form.


A kiss is a lovely trick designed by nature to stop speech when words become superfluous.

URL:

Figure 1-8. *Yahoo! 360 immediately displays the new blast type while still in edit mode*

Overlay Edit

The previous two patterns brought editing inline to the flow of the page. Inline editing keeps the editing in context with the rest of the elements on the page.

Overlay Edit patterns bring the editing form just a layer above the page. While still not leaving the page for editing, it does not attempt to do the editing directly in the flow of the page. Instead a lightweight pop-up layer (e.g., dialog) is used for the editing pane.

There are several reasons for choosing **Overlay Edit** instead of **Inline Edit**.

Sometimes you can't fit a complex edit into the flow of the page. If the editing area is just too large, bringing editing inline can shuffle content around on the page, detracting from the overall experience. A noisy transition from display to edit mode is not desirable.

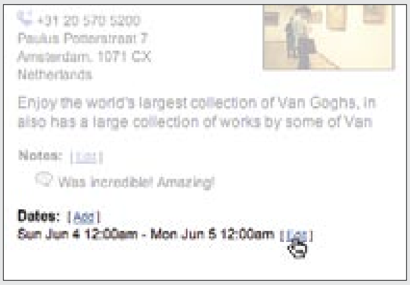
* What You See Is What You Get: an interface where content displayed during editing appears very similar to the final output.

At other times you might choose to interrupt the flow, especially if the information being edited is important in its own right. Overlays give the user a definite editing space. A lightweight overlay does this job nicely.*

Tip

An **Overlay Edit** is a good choice if the editing pane needs dedicated screen space and the context of the page is not important to the editing task.

Yahoo! Trip Planner is an online place for creating and sharing trip plans. Trips contain itinerary items that can be scheduled. When scheduled, the itinerary contains the dates the item is scheduled. Each item can be edited in an overlay (Figure 1-9).



+31 20 570 5200
Paulus Potterstraat 7
Amsterdam, 1071 CX
Netherlands

Enjoy the world's largest collection of Van Goghs, in
also has a large collection of works by some of Van

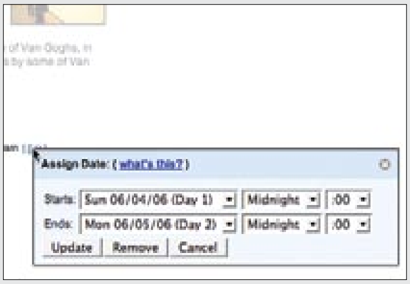
Notes: [Edit]
Was incredible! Amazing!

Dates: [Add]
Sun Jun 4 12:00am - Mon Jun 5 12:00am [Edit]

Non-editing state

Each itinerary item in Yahoo! Trip Planner can be scheduled.

You can add dates or edit an existing one. The dates scheduled are displayed in a simple readable format.



Assign Date: (what's this?)

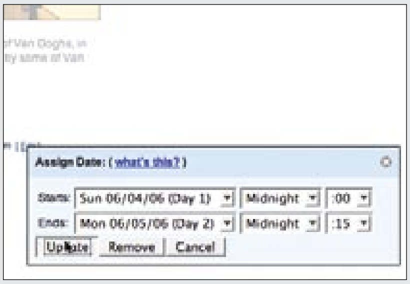
Starts: Sun 06/04/06 (Day 1) | Midnight | :00
Ends: Mon 06/05/06 (Day 2) | Midnight | :00

Invitation to edit

The "[Edit]" link is the invitation to edit.

It is shown at all times.

* In the past, separate browser windows were used for secondary windows. Lightweight overlays simply map the secondary content into a floating layer on the page. The resulting overlay feels more lightweight. See Chapter 5.



Assign Date: (what's this?)

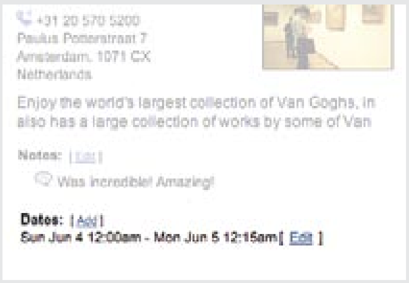
Starts: Sun 06/04/06 (Day 1) | Midnight | :00

Ends: Mon 06/05/06 (Day 2) | Midnight | :15

Editing

The edit form is shown in an overlay.

Since scheduling an itinerary item requires several fields of input, the overlay provides a nice place to contain this editing.



+31 20 570 5200
Paulus Potterstraat 7
Amsterdam, 1071 CX
Netherlands

Enjoy the world's largest collection of Van Goghs, in
also has a large collection of works by some of Van

Notes: [Edit]

Was incredible! Amazing!

Dates: [Add]
Sun Jun 4 12:00am - Mon Jun 5 12:15am [Edit]

Completion

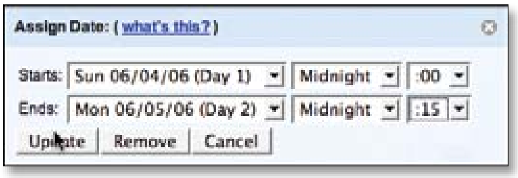
The itinerary item is scheduled after pressing the Update button.

No transitions are used. The presence of the scheduled date signifies that it has been added.

Figure 1-9. *Yahoo! Trip Planner provides a complex editor in an overlay for scheduling an itinerary item*

Considerations

"Sun Jun 4 12:00am—Mon Jun 5 12:00am" is easier to read than a format appropriate for editing (Figure 1-10). Using an editor prevents errors when entering the start and end dates for a specific itinerary item.



Assign Date: (what's this?)

Starts: Sun 06/04/06 (Day 1) | Midnight | :00

Ends: Mon 06/05/06 (Day 2) | Midnight | :15

Figure 1-10. *Yahoo! Trip Planner provides an overlay editor for adjusting itinerary times*

Since the range of dates is known, Trip Planner uses a series of drop-downs to pick the start and end dates along with the time.

It should be noted that using multiple drop-downs for choosing the hour and minute is not the best experience. Although not in the context of an overlay, a better example of choosing an event time when creating an event can be found at Upcoming.org (Figure 1-11).

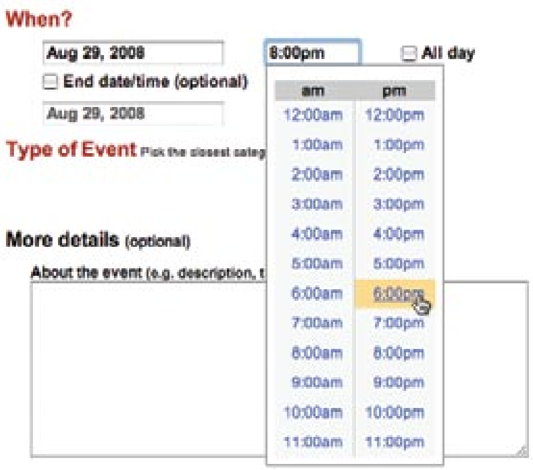


Figure 1-11. *Upcoming provides a better experience for choosing time of day*

The experience of picking a time from a single list (or typing the time in) is more direct than navigating multiple drop-downs.

Why an overlay?

An overlay should be considered when:

- The editing module is considerably larger than the display values.
- Opening an area on the page for the editing module would be distracting or push important information down the page.
- There is concern that the opened information might go partially below the fold. An overlay can be positioned to always be visible in the page.
- You want to create a clear editing area for the user.
- What you are editing is not frequently edited. Having to click on an edit link, adjust to the pop-up location, perform your edit, and close the dialog is a tedious way to edit a series of items. In such cases, opt to either dedicate a space on the page for each item as it is selected, or allow the editing to occur in context to remove some of the time required to deal with an overlay.
- What you are editing is a single entity. If you have a series of items, you should not obscure the other similar items with an overlay. By allowing the edit to occur in context, you can see what the other item's values are while editing.

Best Practices for Inline Edit and Overlay Edit

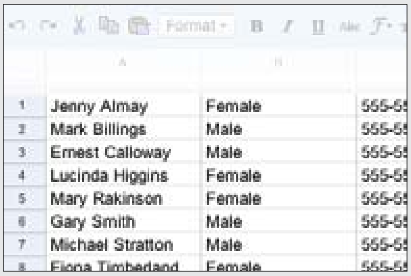
In-Page Editing provides a nice way to change displayed content and observe the change in context. Here are some best practices to consider:

- Keep the editing inline for single fields.
- Use inline when editing one of many in a set. This keeps the context in view.
- Keep the display and editing modes the same size when possible. This will avoid page jitter and reduce distraction when moving between the two modes.
- Make the transition between display and editing as smooth as possible.
- Use mouse hover invitations to indicate editing when readability is primary.
- Avoid using double-click to activate editing.
- Place a bracketed "[edit]" link near the item to be edited if editability is equally important or if the quantity of items that can be edited is small. This is a nice way to separate the link from the displayed text without creating visual distractions.
- Show the edit in place when editing one item in a series (to preserve context).
- Use an overlay when what is being edited needs careful attention. This removes the likelihood of accidentally changing a critical value.
- Do not use multiple overlays for additional fields. If you have a complex edit for a series of elements, use one overlay for all.
- When providing an overlay, use the most lightweight style available to reduce the disruptiveness of the context switch between render and editing state.
- Use buttons when it might be too subtle to trigger completion implicitly.
- Use explicit buttons for saving and canceling when there is room.
- Whenever possible, allow the overlay to be draggable so that obscured content can be revealed as needed.

Table Edit

Editing tables of data is less common in consumer web applications. In enterprise web applications, however, tables reign supreme. The most common request is for the table editing to work like Microsoft Excel, which long ago set the standard for editing data in a grid.

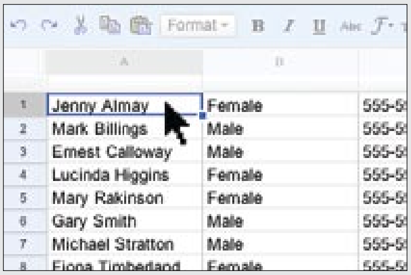
A good example of **Table Edit** is a Google Docs Spreadsheet (Figure 1-12).



	A	B	
1	Jenny Almay	Female	555-5
2	Mark Billings	Male	555-5
3	Ernest Calloway	Male	555-5
4	Lucinda Higgins	Female	555-5
5	Mary Rakinson	Female	555-5
6	Gary Smith	Male	555-5
7	Michael Stratton	Male	555-5
8	Fiona Timberland	Female	555-5

Non-editing state

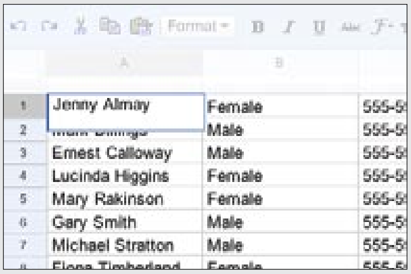
Each cell is tuned toward readability. There are no editing clues while in this state.



	A	B	
1	Jenny Almay	Female	555-5
2	Mark Billings	Male	555-5
3	Ernest Calloway	Male	555-5
4	Lucinda Higgins	Female	555-5
5	Mary Rakinson	Female	555-5
6	Gary Smith	Male	555-5
7	Michael Stratton	Male	555-5
8	Fiona Timberland	Female	555-5

Invitation to edit

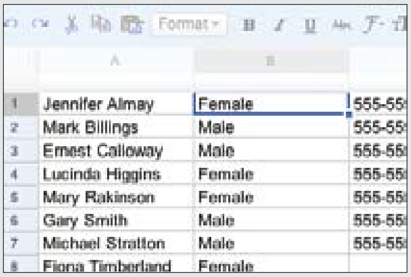
The invitation to edit comes after the cell is clicked.



	A	B	
1	Jenny Almay	Female	555-5
2		Male	555-5
3	Ernest Calloway	Male	555-5
4	Lucinda Higgins	Female	555-5
5	Mary Rakinson	Female	555-5
6	Gary Smith	Male	555-5
7	Michael Stratton	Male	555-5
8	Fiona Timberland	Female	555-5

Editing

An edit box is overlaid on top of the cell being edited. This invites the user to actually type as much text as she needs into the cell.



	A	B	
1	Jennifer Almay	Female	555-55
2	Mark Billings	Male	555-55
3	Ernest Calloway	Male	555-55
4	Lucinda Higgins	Female	555-55
5	Mary Rakinson	Female	555-55
6	Gary Smith	Male	555-55
7	Michael Stratton	Male	555-55
8	Fiona Timberland	Female	

Completion

Completion occurs when the user tabs, clicks outside the cell, or presses Enter.

Figure 1-12. *Editing a spreadsheet in Google Docs is very similar to editing a spreadsheet in Microsoft Excel*

Considerations

Presentation is the primary consideration when displaying a table of data. Editing is secondary. As a result, the editing scaffolding is hidden and only revealed when it's clear the user wants to edit a cell.

Activation

A single mouse click is required to start editing a cell instead of a mouse hover. This is consistent with keeping the display of the grid uncluttered. Imagine how irritating it would be if every mouse motion revealed an edit box.

Tip

You should generally avoid double-click in web applications. However, when web applications look and behave like desktop applications, double-click can be appropriate.

Rendering versus editing. Google Spreadsheet displays the edit box slightly larger than the cell. This clearly indicates editability and lets the user know that input is not limited to the size of the cell (the edit box actually dynamically resizes as the user types into it). The only issue to consider is that the larger edit area covers other table cells. However, this works well in this case since editing is explicitly triggered by a mouse click. If activation had occurred on a mouse hover, the edit mode would have interfered with cell-to-cell navigation.

Best Practices for Table Edit

Here are some best practices for **Table Edit**:

- Bias the display toward readability of the table data.
- Avoid mouse hover for activating cell editing. It makes for the feeling of "mouse traps" and makes the interaction noisy.
- Activate edit with a single click. While using a double-click may not be totally unexpected (since it looks like an Excel spreadsheet), a single click is easier to perform.
- Consider allowing extra space during editing either through a drop-down editor or by slightly enlarging the edit cell.
- As much as possible, mimic the normal conventions of cell navigation that users will already be familiar with (e.g., in Microsoft Excel).

Group Edit

As mentioned before, it is a good idea to keep the differences between the edit mode and the display mode as minimal as possible. In fact, it is a good idea to minimize modes where possible. In honor of this principle, a former manager of mine sported a vanity plate with the phrase "NOMODES". However, modes cannot be avoided altogether, as they do provide necessary context for completing specific tasks.

If you want to keep the display of items on the page as uncluttered as possible while still supporting editing, consider using a single mechanism to enter a special editing mode: **Group Edit**.

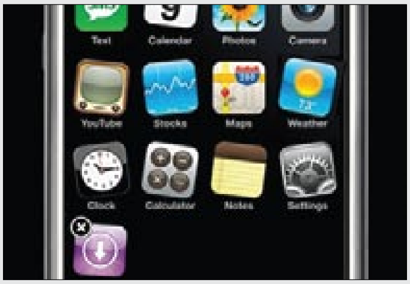
On the iPhone's home screen, the icons are normally locked down. However, there is a way to switch into a special **Group Edit** mode that allows you to rearrange the icon's positions by drag and drop. You enter the mode by pressing down continuously on an icon until the editing mode is turned on (Figure 1-13).

Non-editing state

Tapping icons on the iPhone launches an application.

What we would like is to switch into a mode where we can move the icons around.

Holding down on an icon activates the rearrange mode.



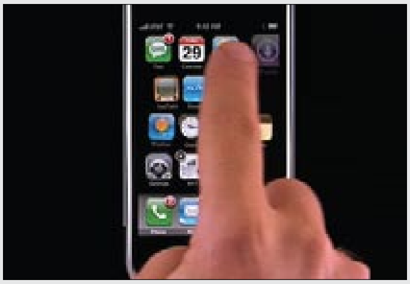
Invitation to rearrange

When the mode is activated, all of the icons begin to wiggle.

This gives the clue that the icons are "loose" and may be rearranged.

Rearranging

Tapping an icon no longer launches an application. Instead, you can drag and drop icons into new positions.



Completion

Pressing the "Home" button returns you to the normal Home screen functionality (exiting the rearranging mode).

Figure 1-13. *The iPhone has a special mode for rearranging applications on the home page—pressing and holding down on an icon places all the applications in "wiggly mode"*

Considerations

The Apple technique signifies that we have entered a special editing mode. When the icons become "wiggly," it is not a large intuitive leap that the icons have become loose and thus we can rearrange them.

Discoverability

Admittedly, the feature is not very discoverable. But it can be argued that it is straightforward once discovered. However, pressing the home button deactivates the rearranging mode. This really should operate more like a toggle. A better way to exit the "wiggly" mode would be to press and hold down on a wiggly icon. It follows the idea that you are pushing the icon back into its fixed place. Since deactivation is not the same mechanism as activation, it is a little hard to figure out how to go back into the normal display mode.

Tip

Activation and deactivation should normally follow the same interaction style. This makes it easy to discover the inverse action. This is a principle we call *Symmetry of Interaction*.

Another example of group editing is in the 37 Signals product, Basecamp (Figure 1-14). When sharing files with Basecamp, you can organize them into various categories. The categories are like folders. Clicking on a category link shows all the files in that "folder." What if you want to delete a category? Or rename it? At the top of the category section there is a single "Edit" link that turns on editing for the whole area.

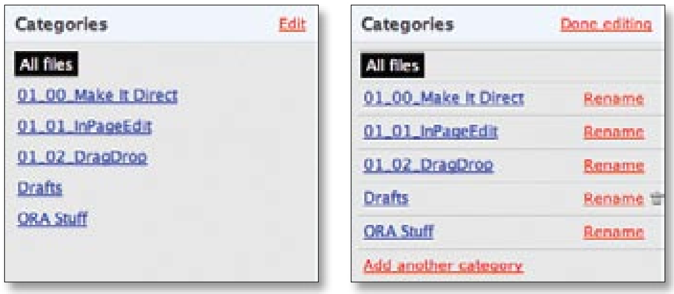


Figure 1-14. *37 Signals Basecamp provides a way to toggle a set of items into edit mode*

Once the **Group Edit** mode is entered, you can add another category, rename an existing category, or delete empty categories. Notice the "Edit" link toggled to read "Done Editing". Clicking this link exits the group-editing mode.

Tip

Switching between edit modes should happen instantaneously. There is no point in making the user wait on an animation to finish before he can start editing.

Discoverability versus readability

The advantage of providing a toggling edit mode is that it keeps the display uncluttered with the editing scaffolding. The disadvantage is that it is less discoverable. This tension between discoverability versus readability is common and must be balanced by the needs of the user.

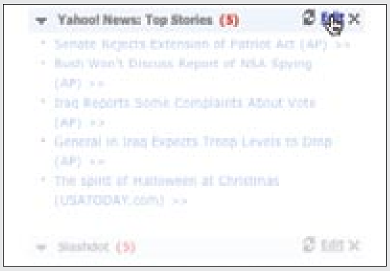
Symmetry of Interaction

Unlike the iPhone example, you turn off editing in the same manner and location that you switched it on. The "Done Editing" link is in the same spot as the "Edit" link was. Since both are hyperlinks, they have the same interaction style. Interactions should be symmetrical wherever possible.

Module Configuration

Popular portal sites like Yahoo! and Google's interactive home page display specific content modules (e.g., Top Stories).

Module Configuration is a common pattern on these types of sites. Instead of modifying modules on a separate page, the sites provide ways to directly configure the amount and type of content that shows in each module. The My Yahoo! home page provides an "Edit" link that allows for **Module Configuration** (Figure 1-15).



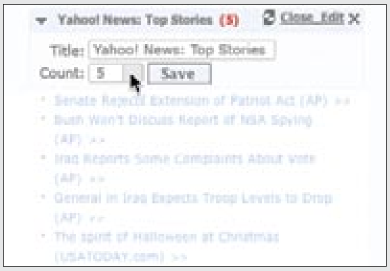
Yahoo! News: Top Stories (5) Edit

- Senate Rejects Extension of Patriot Act (AP) >>
- Bush Won't Discuss Report of NSA Spying (AP) >>
- Iraq Reports Some Complaints About Vote (AP) >>
- General in Iraq Expects Troop Levels to Drop (AP) >>
- The spirit of Halloween at Christmas (USATODAY.com) >>

Slashdot (5) Edit

Non-configuration state

A news module contains top news stories. Each module also has an "Edit" link.



Yahoo! News: Top Stories (5) Close Edit

Title: Yahoo! News: Top Stories

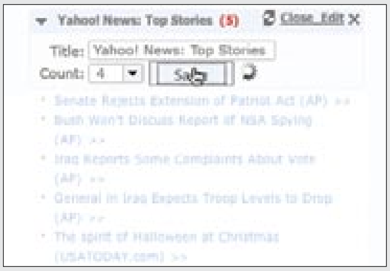
Count: 5 Save

- Senate Rejects Extension of Patriot Act (AP) >>
- Bush Won't Discuss Report of NSA Spying (AP) >>
- Iraq Reports Some Complaints About Vote (AP) >>
- General in Iraq Expects Troop Levels to Drop (AP) >>
- The spirit of Halloween at Christmas (USATODAY.com) >>

Invitation to configure

The "Edit" link is the invitation to edit.

Activating edit slides in a mini-configuration panel in context with the news stories.



Yahoo! News: Top Stories (5) Close Edit

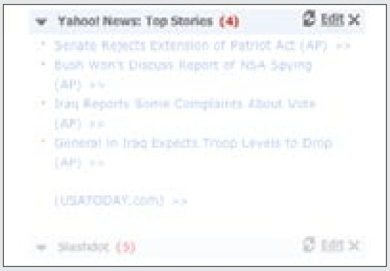
Title: Yahoo! News: Top Stories

Count: 4 Save

- Senate Rejects Extension of Patriot Act (AP) >>
- Bush Won't Discuss Report of NSA Spying (AP) >>
- Iraq Reports Some Complaints About Vote (AP) >>
- General in Iraq Expects Troop Levels to Drop (AP) >>
- The spirit of Halloween at Christmas (USATODAY.com) >>

Configuration

The mini-form allows you to change the title and the number of stories that show. You can save the changes with the "Save" button. The "Close Edit" link forms a symmetrical interaction for edit activation and deactivation. However, there is ambiguity: does "Close Edit" do the same operation as "Save"?


Yahoo! News: Top Stories (4)
Edit
Senate Rejects Extension of Patriot Act (AP) »
Bush Won't Discuss Report of NSA Spying (AP) »
Iraq Reports Some Complaints About Vote (AP) »
General in Iraq Expects Troop Levels to Drop (AP) »
(USATODAY.com) »
Slashdot (5)
Edit

Completion

The number of news stories changes and the mini-configuration panel slides back out of the way.

Figure 1-15. *Configuring modules on the My Yahoo! page can be done directly in place*

Considerations

There are some issues to consider when using **Module Configuration**.

Visual noise

Putting edit links on each module can be visually noisy. An alternative approach is to use the **Group Edit** pattern (as we saw in Figure 1-14) to place an edit link at the page level that turns on edit links for each module. When the "Done Editing" link is clicked, the links for each module are hidden. Again the trade-off is between visual noise and discoverability.

Best Practices for Group Edit and Module Configuration

Here are some best practices to keep in mind:

- Use an edit toggle when there are a number of items to edit and showing edit scaffolding would make the display visually noisy.
- Make activation and deactivation as similar as possible (Symmetry of Interaction). Switching in and out of an editing mode should operate more like a toggle.
- Provide inline edit configuration for modules when configuration is an important feature.
- Provide a way to turn configuration on/off globally for module configuration when this is secondary to content display.

Guidelines for Choosing Specific Editing Patterns

In-Page Edit provides a powerful way to make interfaces direct. Here are some general guidelines to think about when choosing an editing pattern:

- Whenever you have a single field on the page that needs editing, consider using the **Single-Field Inline Edit**.
- For multiple fields or more complex editing, use the **Multi-Field Inline Edit**.
- If you don't need inline context while editing, or the editing is something that demands the user's full attention, use **Overlay Edit**.
- For grid editing, follow the pattern **Table Edit**.
- When dealing with multiple items on a page, **Group Edit** provides a way to balance between visual noise and discoverability.
- When providing direct configuring to modules, use the **Module Configuration** pattern.

Chapter 2

Drag and Drop

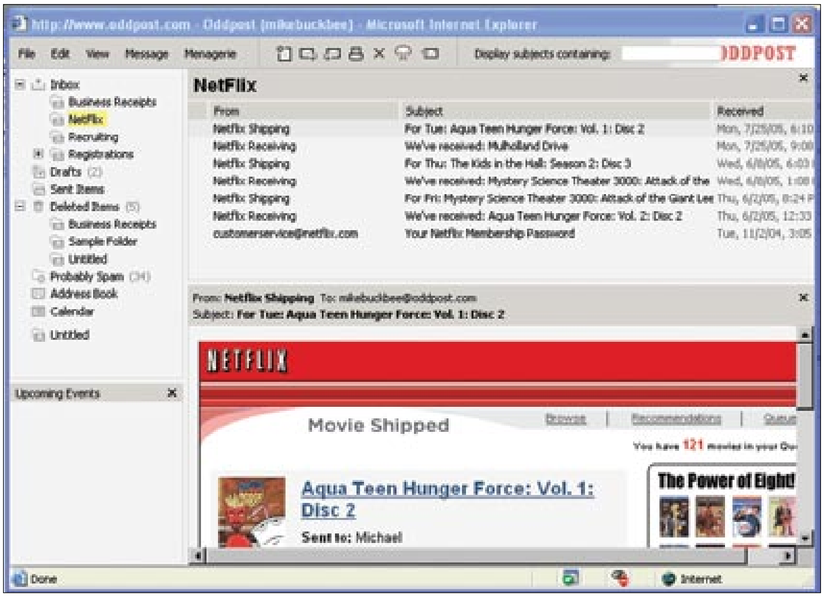
One of the great innovations that the Macintosh brought to the world in 1984 was **Drag and Drop**. Influenced by the graphical user interface work on Xerox PARC's Star Information System and subsequent lessons learned from the Apple Lisa, the Macintosh team invented drag and drop as an easy way to move, copy, and delete files on the user's desktop.

It was quite a while before drag and drop made its way to the Web in any serious application. In 2000, a small startup, HalfBrain,[*] launched a web-based presentation application, BrainMatter. It was written entirely in DHTML and used drag and drop as an integral part of its interface.

Drag and drop showed up again with another small startup, Oddpost,[†] when it launched a web-based mail application (Figure 2-1) that allowed users to drag and drop messages between folders.

* HalfBrain also created a full spreadsheet application written in DHTML prior to this. It included many of the features of Microsoft Excel.

† Some of the developers at Oddpost actually came from HalfBrain. Yahoo! later purchased Oddpost's mail application to form the basis for the current Yahoo! Mail product.



From	Subject	Received
Netflix Shipping	For Tue: Aqua Teen Hunger Force: Vol. 1: Disc 2	Mon, 7/25/05, 6:10
Netflix Receiving	We've received: Mulholland Drive	Mon, 7/25/05, 9:00
Netflix Shipping	For Thu: The Kids in the Hall: Season 2: Disc 3	Wed, 6/8/05, 6:03
Netflix Receiving	We've received: Mystery Science Theater 3000: Attack of the	Wed, 6/8/05, 1:08
Netflix Shipping	For Fri: Mystery Science Theater 3000: Attack of the Giant Lee	Thu, 6/2/05, 8:24
Netflix Receiving	We've received: Aqua Teen Hunger Force: Vol. 2: Disc 2	Thu, 6/2/05, 12:33
customerservice@netflix.com	Your Netflix Membership Password	Tue, 11/2/04, 3:05

From: **Netflix Shipping** To: mikebuckbee@oddpost.com
Subject: **For Tue: Aqua Teen Hunger Force: Vol. 1: Disc 2**

Figure 2-1. *The Oddpost web mail client performed like a desktop mail application and included drag and drop as a key feature*

The biggest hindrance was the difficulty in saving the user's state after a drag was completed without refreshing the page. It was possible, but the underlying technology was not consistent across all browsers. Now that the technologies underlying Ajax* have become widely known and a full complement of browsers support these techniques, **Drag and Drop** has become a more familiar idiom on the Web.

Interesting Moments

At first blush, drag and drop seems simple. Just grab an object and drop it somewhere. But, as always, the devil is in the details. There are a number of individual states at which interaction is possible. We call these microstates *interesting moments*:†

- How will users know what is draggable?
- What does it mean to drag and drop an object?
- Where can you drop an object, and where is it not valid to drop an object?
- What visual affordance will be used to indicate draggability?

* Jesse James Garrett, founder of Adaptive Path, originally defined the term *Ajax* as "Asynchronous JavaScript and XML." However, it generally means being able to retrieve information or save state without refreshing the page—an integral ability for any rich interaction.

† Bill Scott, author of this book, originally called these *interaction events*. Eric Miraglia, a former colleague of his at Yahoo!, coined the more colorful term *interesting moments*.

- During drag, how will valid and invalid drop targets be signified?
- Do you drag the actual object?
- Or do you drag just a ghost of the object?
- Or is it a thumbnail representation that gets dragged?
- What visual feedback should be used during the drag and drop interaction?

What makes it challenging is that there are a lot of events during drag and drop that can be used as opportunities for feedback to the user. Additionally, there are a number of elements on the page that can participate as actors in this feedback loop.

The Events

There are at least 15 events available for cueing the user during a drag and drop interaction:

Page Load
Before any interaction occurs, you can pre-signify the availability of drag and drop. For example, you could display a tip on the page to indicate draggability.

Mouse Hover
The mouse pointer hovers over an object that is draggable.

Mouse Down
The user holds down the mouse button on the draggable object.

Drag Initiated
After the mouse drag starts (usually some threshold—3 pixels).

Drag Leaves Original Location
After the drag object is pulled from its location or object that contains it.

Drag Re-Enters Original Location
When the object re-enters the original location.

Drag Enters Valid Target
Dragging over a valid drop target.

Drag Exits Valid Target
Dragging back out of a valid drop target.

Drag Enters Specific Invalid Target
Dragging over an invalid drop target.

Drag Is Over No Specific Target
Dragging over neither a valid or invalid target. Do you treat all areas outside of valid targets as invalid?

Drag Hovers Over Valid Target
User pauses over the valid target without dropping the object. This is usually when a spring loaded drop target can open up. For example, drag over a folder and pause, the folder opens revealing a new area to drag into.

Drag Hovers Over Invalid Target
User pauses over an invalid target without dropping the object. Do you care? Will you want additional feedback as to why it is not a valid target?

Drop Accepted
Drop occurs over a valid target and drop has been accepted.

Drop Rejected
Drop occurs over an invalid target and drop has been rejected. Do you zoom back the dropped object?

Drop on Parent Container
Is the place where the object was dragged from special? Usually this is not the case, but it may carry special meaning in some contexts.

The Actors

During each event you can visually manipulate a number of *actors*. The page elements available include:

- Page (e.g., static messaging on the page)
- Cursor
- Tool Tip
- Drag Object (or some portion of the drag object, e.g., title area of a module)
- Drag Object's Parent Container
- Drop Target

Interesting Moments Grid

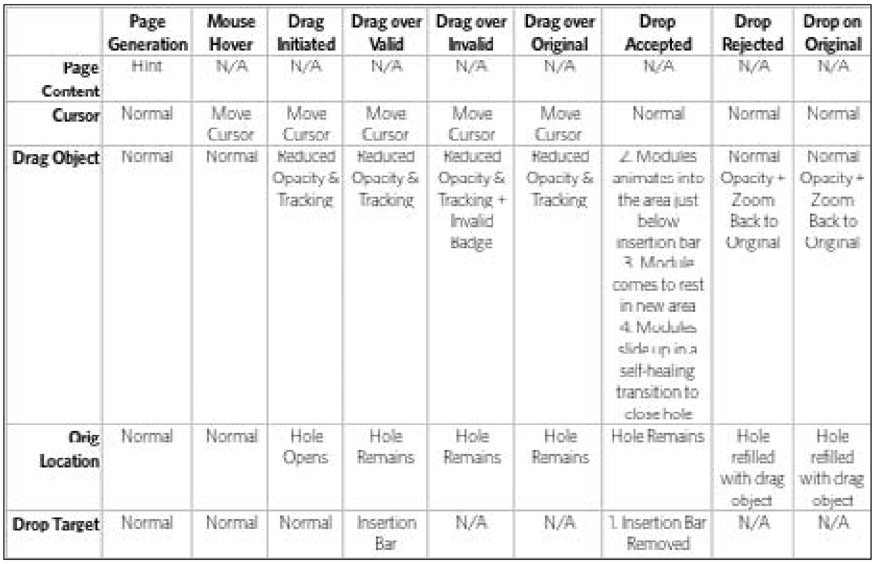
That's 15 events times 6 actors. That means there are 90 possible interesting moments—each requiring a decision involving an almost unlimited number of style and timing choices.

You can pull all this together into a simple interesting moments grid for **Drag and Drop**. Figure 2-2 shows an interesting moments grid for My Yahoo!.

Tip

You can use an interesting moments grid to capture any complex interaction.

The grid is a handy tool for planning out interesting moments during a drag and drop interaction. It serves as a checklist to make sure there are no "holes" in the interaction. Just place the actors along the lefthand side and the moments along the top. In the grid intersections, place the desired behaviors.



	Page Generation	Mouse Hover	Drag Initiated	Drag over Valid	Drag over Invalid	Drag over Original	Drop Accepted	Drop Rejected	Drop on Original
Page Content	Hint	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Cursor	Normal	Move Cursor	Move Cursor	Move Cursor	Move Cursor	Move Cursor	Normal	Normal	Normal
Drag Object	Normal	Normal	Reduced Opacity & Tracking	Reduced Opacity & Tracking	Reduced Opacity & Tracking + Invalid Badge	Reduced Opacity & Tracking	2. Modules animates into the area just below insertion bar 3. Module comes to rest in new area 4. Modules slide up in a self-healing transition to close hole	Normal Opacity + Zoom Back to Original	Normal Opacity + Zoom Back to Original
Orig Location	Normal	Normal	Hole Opens	Hole Remains	Hole Remains	Hole Remains	Hole Remains	Hole refilled with drag object	Hole refilled with drag object
Drop Target	Normal	Normal	Normal	Insertion Bar	N/A	N/A	1. Insertion Bar Removed	N/A	N/A

Figure 2-2. *A simplified interesting moments grid for the original My Yahoo! drag and drop design;*[*] *it provided a way to capture the complexities of drag and drop into a single page*

Purpose of Drag and Drop

Drag and drop can be a powerful idiom if used correctly. Specifically it is useful for:

Drag and Drop Module
: Rearranging modules on a page.

Drag and Drop List
: Rearranging lists.

Drag and Drop Object
: Changing relationships between objects.

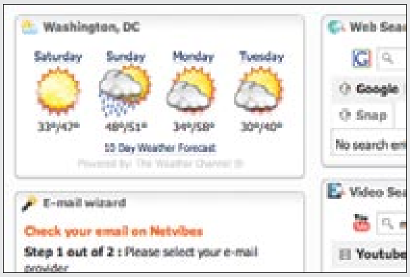
Drag and Drop Action
: Invoking actions on a dropped object.

Drag and Drop Collection
: Maintaining collections through drag and drop.

[*] A template for the interesting moments grid can be found at *http://designingwebinterfaces.com/resources/interestingmomentsgrid.xls*.

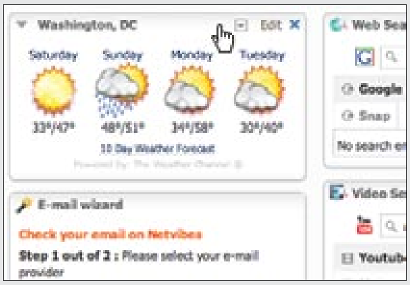
Drag and Drop Module

One of the most useful purposes of drag and drop is to allow the user to directly place objects where she wants them on the page. A typical pattern is **Drag and Drop Modules** on a page. Netvibes provides a good example of this interaction pattern (Figure 2-3).



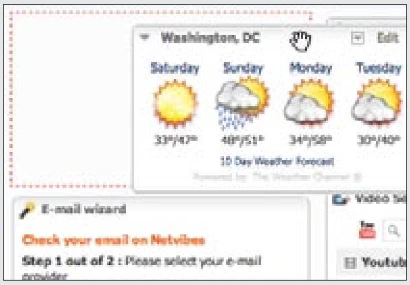
Normal display style

Modules are displayed without an explicit cue for drag and drop.



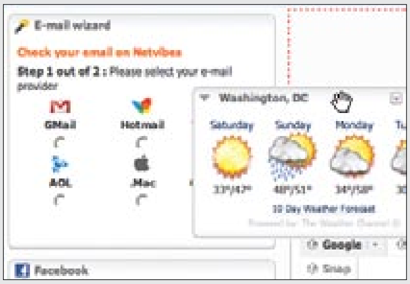
Invitation to drag

Moving the mouse to a module's header changes the cursor to indicate that the item is draggable.



Dragging

The module being moved is dragged directly. A ripped-out "hole" is exposed where the module was dragged from.



Invitation to drop

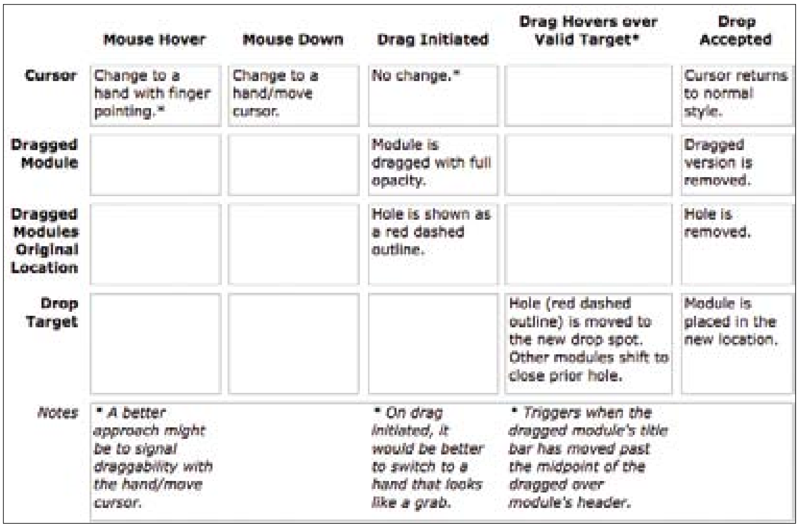
Dragging the module opens up a new hole indicating where the object will be dropped.

The hole always indicates where the object will go when dropped.

Figure 2-3. *Netvibes allows modules to be arranged directly via drag and drop; the hole cues what will happen when a module is dropped*

Considerations

Netvibes allows its modules to be rearranged with drag and drop. A number of interesting moments decide the specific interaction style for this site. Figure 2-4 shows the interesting moments grid for Netvibes.



	Mouse Hover	Mouse Down	Drag Initiated	Drag Hovers over Valid Target*	Drop Accepted
Cursor	Change to a hand with finger pointing.*	Change to a hand/move cursor.	No change.*		Cursor returns to normal style.
Dragged Module			Module is dragged with full opacity.		Dragged version is removed.
Dragged Modules Original Location			Hole is shown as a red dashed outline.		Hole is removed.
Drop Target				Hole (red dashed outline) is moved to the new drop spot. Other modules shift to close prior hole.	Module is placed in the new location.
Notes	** A better approach might be to signal draggability with the hand/move cursor.*		** On drag initiated, it would be better to switch to a hand that looks like a grab.*	** Triggers when the dragged module's title bar has moved past the midpoint of the dragged over module's header.*	

Figure 2-4. *Interesting moments grid for Netvibes: there are 20 possible moments of interaction; Netvibes specifically handles 9 of these moments*

While dragging, it is important to make it clear what will happen when the user drops the dragged object. There are two common approaches to targeting a drop:

- Placeholder target
- Insertion target

Placeholder target

Netvibes uses a placeholder (hole with dashed outline) as the drop target. The idea (illustrated in Figure 2-5) is to always position a hole in the spot where the drop would occur. When module ① starts dragging, it gets "ripped" out of the spot. In its place is the placeholder target (dashed outline). As ① gets dragged to the spot between ③ and ④, the placeholder target jumps to fill in this spot as ④ moves out of the way.

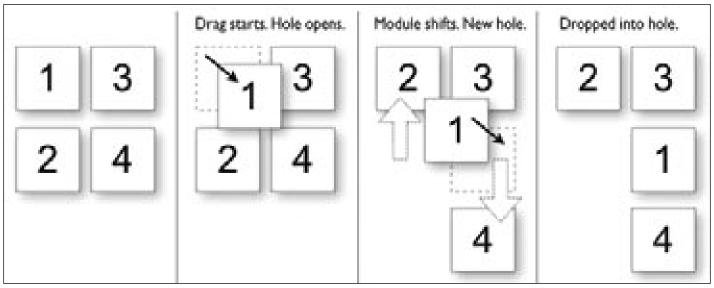


Figure 2-5. *A placeholder target always shows where the dragged module will end after the drop; module 1 is being dragged from the upper right to the position between modules 3 and 4*

The hole serves as a placeholder and always marks the spot that the dragged module will land when dropped. It also previews what the page will look like (in relation to the other modules) if the drop occurs there. For module drag and drop, the other modules only slide up or down within a vertical column to make room for the dragged module.

One complaint with using placeholder targets is that the page content jumps around a lot during the drag. This makes the interaction noisier and can make it harder to understand what is actually happening. This issue is compounded when modules look similar. The user starts dragging the modules around and quickly gets confused about what just got moved. One way to resolve this is to provide a quick animated transition as the modules move. It is important, however, that any animated transitions not get in the way of the normal interaction. In Chapter 11, we will discuss timing of transitions in detail.

There is a point in Figure 2-5 where the placeholder shifts to a new location. What determines placeholder targeting? In other words, what determines where the user is intending to place the dragged object? The position of the mouse, the boundary of the dragged object, and the boundary of the dragged-over object can all be used to choose the module's new location.

Boundary-based placement. Since most sites that use placeholder targeting drag the module in its original size, targeting is determined by the boundaries of the dragged object and the boundaries of the dragged-over object. The mouse position is usually ignored because modules are only draggable in the title (a small region). Both Netvibes and iGoogle take the boundary-based approach. But, interestingly, they calculate the position of their placeholders differently.

In Netvibes, the placeholder changes position only after the dragged module's title bar has moved beyond the dragged-over module's title bar. In practice, this means if you are moving a small module to be positioned above a large module, you have to move it to the very top of the large module. In Figure 2-6 you have to drag the small "To Do List" module all the way to the top of the "Blog Directory" module before the placeholder changes position.

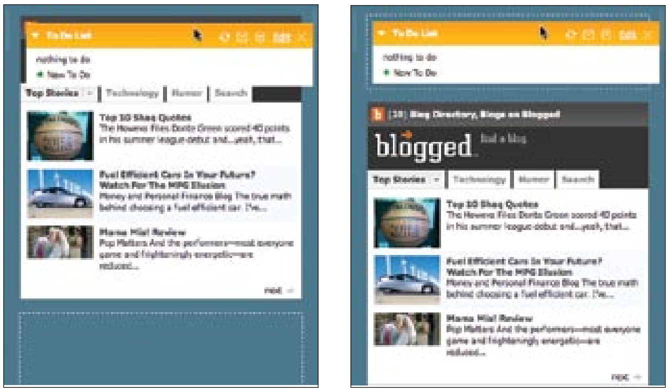


Figure 2-6. *In Netvibes, dragging a small module to be placed above a large module requires dragging a large distance; the "To Do List" has to be dragged to the top of the "Blog Directory" module*

In contrast, moving the small module below the large module actually requires less drag distance since you only have to get the title bar of the small module below the title bar of the large module (Figure 2-7).

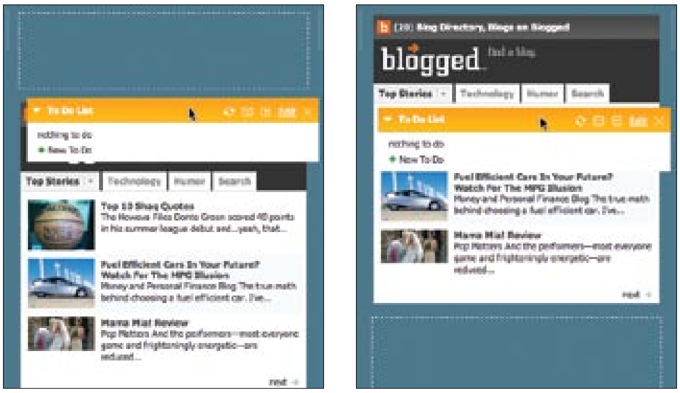


Figure 2-7. *Dragging a small module below a large module requires a smaller drag distance; since the targeting is based on the header of the dragged-over module, the drag distance in this scenario is less than in the previous figure*

This approach to boundary-based drop targeting is non-symmetrical in the drag distance when dragging modules up versus dragging modules down (Figure 2-8).

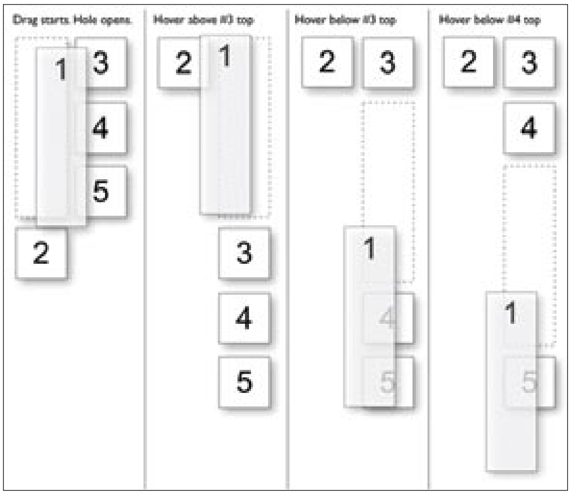


Figure 2-8. *The Netvibes approach requires the dragged object's title to be placed above or below a module before the placement position changes; this results in inconsistent drag distances*

A more desirable approach is that taken by iGoogle. Instead of basing the drag on the *title bar*, iGoogle calculates the placeholder targeting on the dragged-over object's *midpoint*. In Figure 2-9, the stock market module is very large (the module just above the moon phase module).

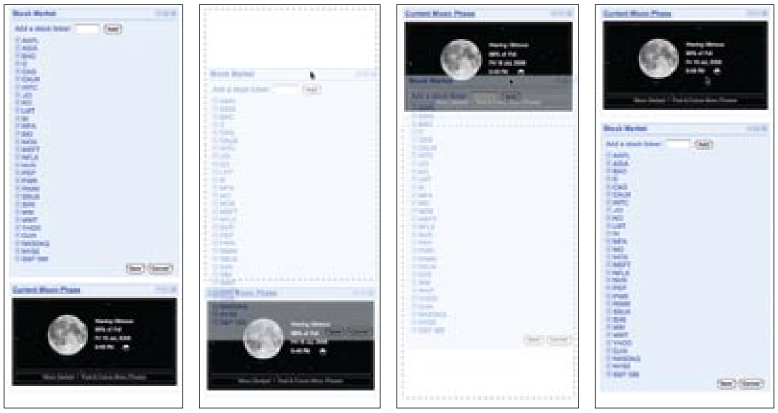


Figure 2-9. *When dragging a module downward, iGoogle moves the placeholder when the bottom of the dragged module crosses the midpoint of the object being dragged over; the distance to accomplish a move is less than in the Netvibes approach*

With the Netvibes approach, you would have to drag the stock module's title below the moon phase module's title. iGoogle instead moves the placeholder when the *bottom* of the dragged module (stock module) crosses the midpoint of the dragged over module (moon phase module).

What happens when we head the other way? When we drag the stock module up to place it above the moon phase module, iGoogle moves the placeholder when the *top* of the stock module crosses the midpoint of the moon phase module (Figure 2-10).

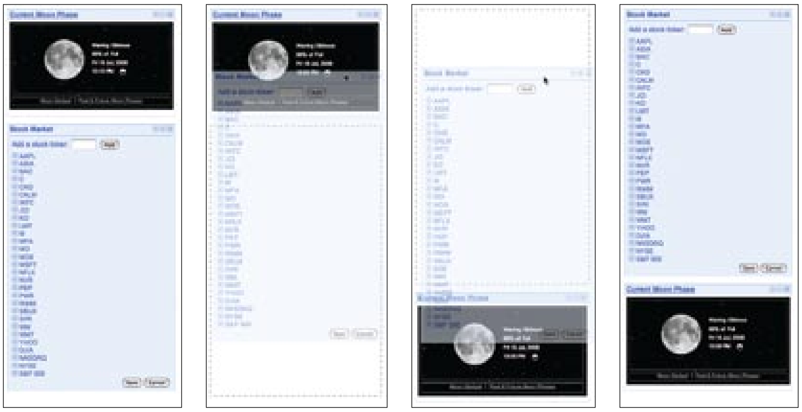


Figure 2-10. *When dragging a module upward, iGoogle moves the placeholder when the top of the dragged module crosses the midpoint of the object being dragged over; dragging modules up or down requires the same effort, unlike in the Netvibes example*

As Figure 2-11 illustrates, module ① is dragged from the first column to the second column, the placeholder moves above module ③. As module ① is dragged downward, the placeholder moves below ③ and ④ as the bottom of module ① crosses their midpoints.

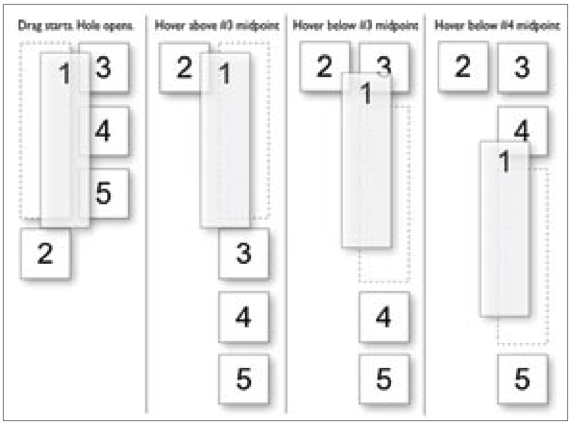
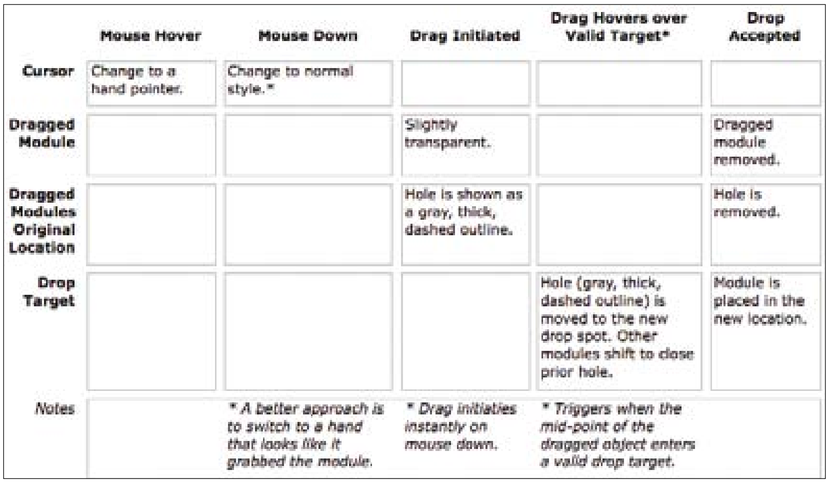


Figure 2-11. *To create the best drag experience, use the original midpoint location of the module being dragged over to determine where to drop the dragged module: module 1 is being dragged into the position just below module 4*

The net result is that the iGoogle approach feels more responsive and requires less mouse movement to position modules. Figure 2-12 shows the interesting moments grid for the iGoogle drag and drop interaction.



	Mouse Hover	Mouse Down	Drag Initiated	Drag Hovers over Valid Target*	Drop Accepted
Cursor	Change to a hand pointer.	Change to normal style.*			
Dragged Module			Slightly transparent.		Dragged module removed.
Dragged Modules Original Location			Hole is shown as a gray, thick, dashed outline.		Hole is removed.
Drop Target				Hole (gray, thick, dashed outline) is moved to the new drop spot. Other modules shift to close prior hole.	Module is placed in the new location.
Notes		** A better approach is to switch to a hand that looks like it grabbed the module.*	** Drag initiaties instantly on mouse down.*	** Triggers when the mid-point of the dragged object enters a valid drop target.*	

Figure 2-12. *Interesting moments grid for iGoogle: as in the Netvibes grid, there are 20 possible moments of interaction; iGoogle specifically handles 8 of these moments*

Insertion target

Placeholder positioning is a common approach, but it is not the only way to indicate drop targeting. An alternate approach is to keep the page as stable as possible and only move around an insertion target (usually an insertion bar). A previous version of My Yahoo! used the insertion bar approach as the dragged module was moved around (see Figure 2-13).

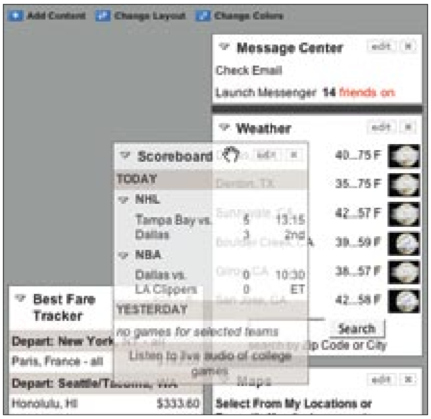


Figure 2-13. *My Yahoo! uses the insertion bar approach*

While the module is dragged, the page remains stable. No modules move around. Instead an insertion bar marks where the module will be placed when dropped.

This technique is illustrated in Figure 2-14. When module ① is dragged to the position between ③ and ④, an insertion bar is placed there. This indicates that if ① is dropped, then ④ will slide down to open up the drop spot.

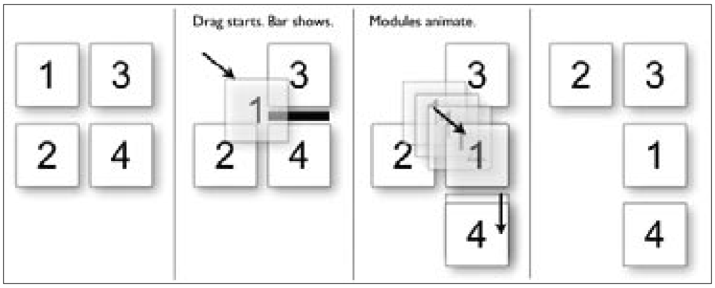
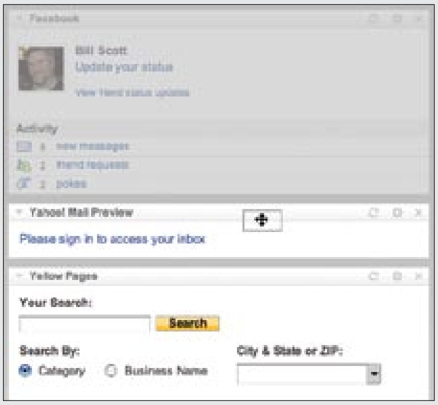


Figure 2-14. *Using an insertion bar keeps the page stable during dragging and makes it clear how things get rearranged when the module is dropped*

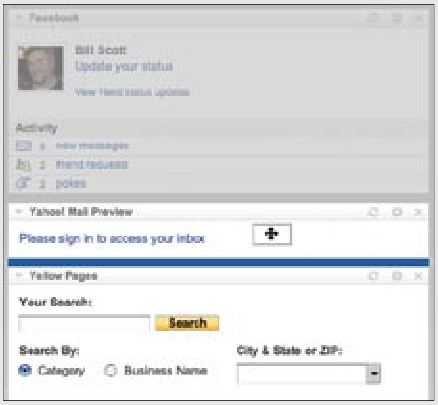
Unlike with the placeholder target, the dragged module ① is usually represented with a slightly transparent version of the module (also known as *ghosting*). This is the approach shown in Figure 2-13 in an earlier version of My Yahoo!. In the most current version, full-size module dragging has been replaced with a thumbnail representation (the small gray outline being dragged in Figure 2-15). This is somewhat unfortunate since the small gray outline is not very visible.



Dragging a module down

Note the gray outline being dragged. The small gray rectangle represents the dragged module (the Facebook module).

Since the drag representation is above the mid-point of the Yahoo! Mail Preview module, no change is indicated for a drop.



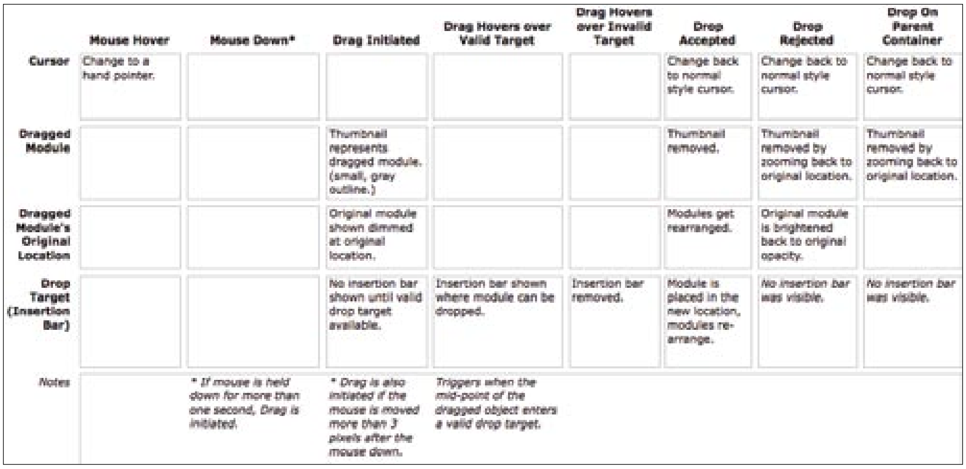
Insertion bar appears

The dragged module representation is now below the Yahoo! Mail Preview module's midpoint.

The insertion bar is rendered to show that the Facebook module will be placed just below the Mail Preview module if dropped.

Figure 2-15. *My Yahoo! uses a small gray rectangle to represent the dragged module*

As you can see in Figure 2-16, the My Yahoo! page makes different decisions about how drag and drop modules are implemented as compared to Netvibes (Figure 2-4) and iGoogle (Figure 2-12).



	Mouse Hover	Mouse Down*	Drag Initiated	Drag Hovers over Valid Target	Drag Hovers over Invalid Target	Drop Accepted	Drop Rejected	Drop On Parent Container
Cursor	Change to a hand pointer.					Change back to normal style cursor.	Change back to normal style cursor.	Change back to normal style cursor.
Dragged Module			Thumbnail represents dragged module. (small, gray outline.)			Thumbnail removed.	Thumbnail removed by zooming back to original location.	Thumbnail removed by zooming back to original location.
Dragged Module's Original Location			Original module shown dimmed at original location.			Modules get rearranged.	Original module is brightened back to original opacity.	
Drop Target (Insertion Bar)			No insertion bar shown until valid drop target available.	Insertion bar shown where module can be dropped.	Insertion bar removed.	Module is placed in the new location, modules re-arrange.	*No insertion bar was visible.*	*No insertion bar was visible.*
Notes		** If mouse is held down for more than one second, Drag is initiated.*	** Drag is also initiated if the mouse is moved more than 3 pixels after the mouse down.*	*Triggers when the mid-point of the dragged object enters a valid drop target.*				

Figure 2-16. *My Yahoo! uses 15 of the possible 32 moments to interact with the user during drag and drop; the biggest difference between My Yahoo!, Netvibes, and iGoogle is the insertion bar placement—another subtle difference is how drag gets initiated*

Drag distance

Dragging the thumbnail around does have other issues. Since the object being dragged is small, it does not intersect a large area. It requires moving the small thumbnail directly to the place it will be dropped. With iGoogle, the complete module is dragged. Since the module will always be larger than the thumbnail, it intersects a drop target with much less movement. The result is a shorter drag distance to accomplish a move.

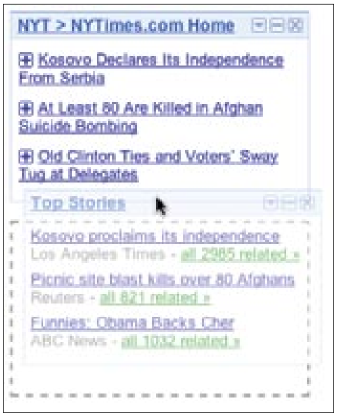
Tip

Keep in mind that **Drag and Drop** takes additional mouse dexterity. If possible, shorten the necessary drag distance to target a drop.

Drag rendering

How should the dragged object be represented? Should it be rendered with a slight transparency (ghost)? Or should it be shown fully opaque? Should a thumbnail representation be used instead?

As shown earlier, My Yahoo! uses a small gray rectangle to represent a module (Figure 2-15). Netvibes represents the dragged module in full size as opaque (shown back in Figure 2-3), while iGoogle uses partial transparency (Figure 2-17). The transparency (ghosting) effect communicates that the object being dragged is actually a representation of the dragged object. It also keeps more of the page visible, thus giving a clearer picture of the final result of a drop.



NYT > NYTimes.com Home

Kosovo Declares Its Independence From Serbia

At Least 80 Are Killed in Afghan Suicide Bombing

Old Clinton Ties and Voters' Sway Tug at Delegates

Top Stories

Kosovo proclaims its independence
Los Angeles Times - all 2985 related »

Picnic site blast kills over 80 Afghans
Reuters - all 821 related »

Funnies: Obama Backs Cher
ABC News - all 1032 related »

Figure 2-17. *On iGoogle the dragged module Top Stories is given transparency to make it easier to see the page and to indicate that we are in a placement mode*

Ghosting the module also indicates that the module is in a special mode. It signals that the module has not been positioned; instead, it is in a transitional state.

Tip

For **Drag and Drop Modules**, use the module's midpoint to control the drop targeting.

Of the various approaches for **Drag and Drop Modules**, iGoogle combines the best approaches into a single interface:

Placeholder targeting
Most explicit way to preview the effect.

Midpoint boundary
Requires the least drag effort to move modules around.

Full-size module dragging
Coupled with placeholder targeting and midpoint boundary detection, it means drag distances to complete a move are shorter.

Ghost rendering
Emphasizes the page rather than the dragged object. Keeps the preview clear.

Best Practices for Drag and Drop Module

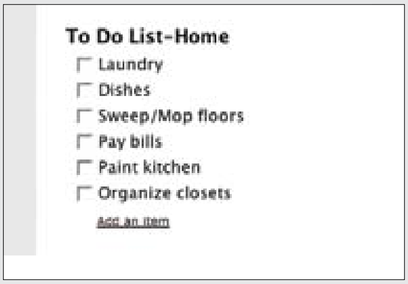
Here are some best practices to keep in mind:

- Use the placeholder approach when showing a clear preview during drag is important.
- Use the insertion bar approach when you want to avoid page jitter.
- Use the midpoint of the dragged object to determine drag position.
- Use a slightly transparent version of the object being dragged (ghost) instead of an opaque version.
- If you drag thumbnail representations, use the insertion bar targeting approach.

Drag and Drop List

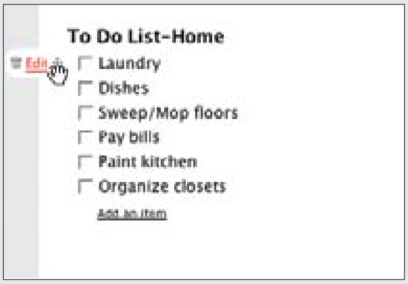
Rearranging lists is very similar to rearranging modules on the page but with the added constraint of being in a single dimension (up/down or left/right). The **Drag and Drop List** pattern defines interactions for rearranging items in a list.

37 Signal's Backpackit allows to-do items to be rearranged with **Drag and Drop List** (Figure 2-18).



Normal display state

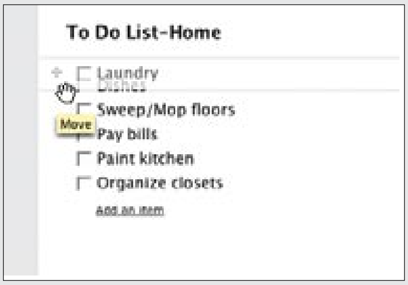
List items are displayed without any indication that the items can be rearranged.



Invitation to drag

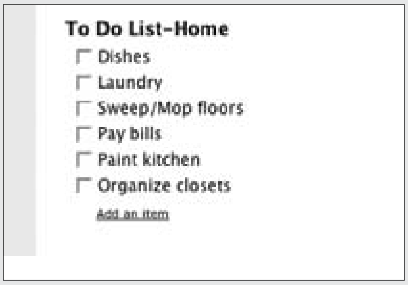
One of the in-context tools revealed during mouse hover shows a four-way arrow indicating that the object can be moved.

Dragging this object allows the list item to be moved.



Dragging

Rearranging occurs in real time. An empty slot is exposed where the dragged item will fit.



Dropped

The item snaps into the new location (where the hole was opened up).

Figure 2-18. *Backpackit allows to-do lists to be rearranged directly via drag and drop*

Considerations

Backpackit takes a real-time approach to dragging items. Since the list is constrained, this is a natural approach to moving objects around in a list. You immediately see the result of the drag.

Placeholder target

This is essentially the same placeholder target approach we discussed earlier for dragging and dropping modules. The difference is that when moving an item in a list, we are constrained to a single dimension. Less feedback is needed. Instead of a "ripped-out" area (represented earlier with a dotted rectangle), a simple hole can be exposed where the object will be placed when dropped.

A good example from the desktop world is Apple's iPhoto. In a slideshow, you can easily rearrange the order of photos with drag and drop. Dragging the photo left or right causes the other photos to shuffle open a drop spot (Figure 2-19).

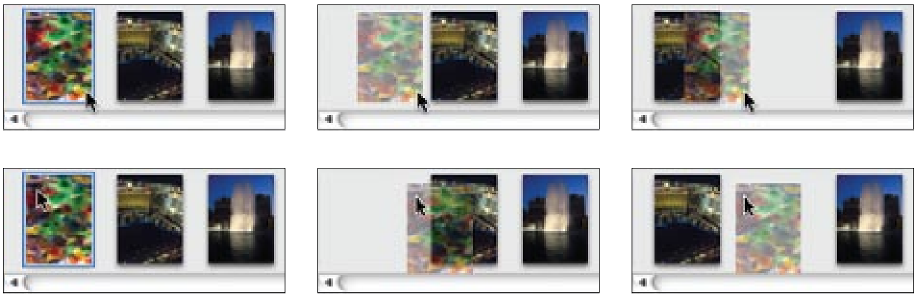


Figure 2-19. *iPhoto uses cursor position: when the cursor crosses a threshold (the edge of the next photo), a new position is opened up*

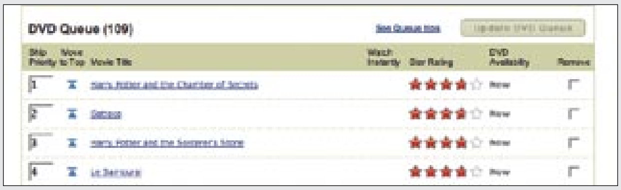
The difference between iPhoto and Backpackit is that instead of using the dragged photo's boundary as the trigger for crossing a threshold, iPhoto uses the mouse cursor position. In the top row of Figure 2-19, the user clicked on the right side of the photo. When the cursor crosses into the left edge of the next photo, a new space is opened. In the bottom row, the user clicked on the top left side of the photo. Notice in both cases it is the mouse position that determines when a dragged photo has moved into the space of another photo, not the dragged photo's boundary.

Tip

In a **Drag and Drop List**, use the mouse position to control where the item will be dropped.

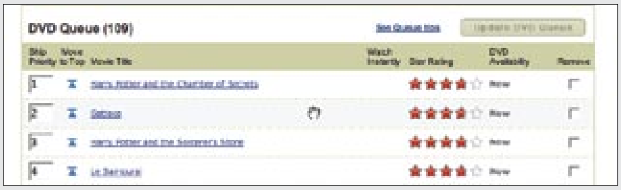
Insertion target

Just as with **Drag and Drop Modules**, placeholder targeting is not the only game in town. You can also use an insertion bar within a list to indicate where a dropped item will land. Netflix uses an *insertion target* when movies are dragged to a new location in a user's movie queue (Figure 2-20).



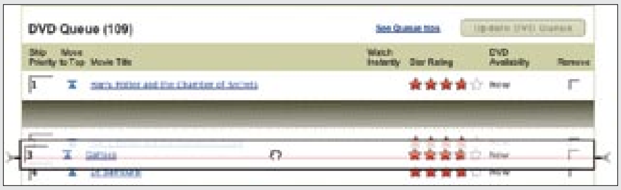
Normal display state

List items are displayed without any indication that the items can be rearranged.



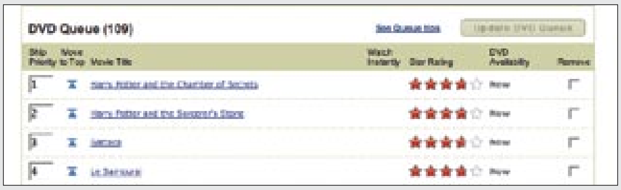
Invitation to drag

The cursor changes to indicate draggability.



Dragging

A hole is marked where the item is pulled from. The dragged item's index number changes and an insertion bar indicates where it will be moved to.



Dropped

The item is moved immediately into the spot marked by the insertion bar.

Figure 2-20. *A Netflix queue can be rearranged via drag and drop*

The upside to this approach is that the list doesn't have to shuffle around during drag. The resulting experience is smoother than the Backpackit approach. The downside is that it is not as obvious where the movie is being positioned. The insertion bar appears under the ghosted item. The addition of the brackets on the left and right of the insertion bar is an attempt to make the targeting clearer.

Non–drag and drop alternative

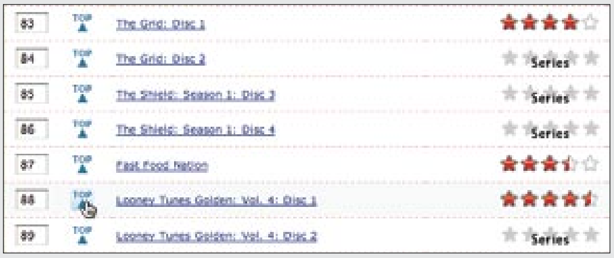
Besides drag and drop, the Netflix queue actually supports two other ways to move objects around:

- Edit the row number and then press the "Update DVD Queue" button.
- Click the "Move to Top" icon to pop a movie to the top.

Modifying the row number is straightforward. It's a way to rearrange items without drag and drop. The "Move to Top" button is a little more direct and fairly straightforward (if the user really understands that this icon means "move to top"). Drag and drop is the least discoverable of the three, but it is the most direct, visual way to rearrange the list. Since rearranging the queue is central to the Netflix customer's satisfaction, it is appropriate to allow multiple ways to do so.

Hinting at drag and drop

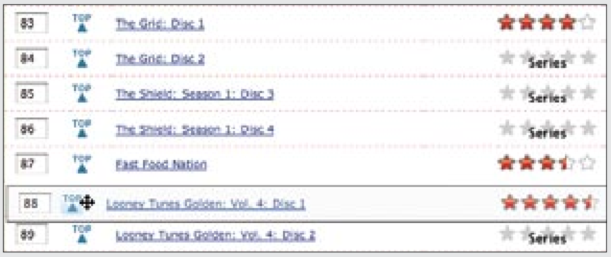
When the user clicks the "Move to Top" button, Netflix animates the movie as it moves up. But first, the movie is jerked downward slightly and then spring-loaded to the top (Figure 2-21).



83	TOP	The Grid: Disc 1	
84	TOP	The Grid: Disc 2	Series
85	TOP	The Shield: Season 1: Disc 3	Series
86	TOP	The Shield: Season 1: Disc 4	Series
87	TOP	Fast Food Nation	
88	TOP	Looney Tunes Golden: Vol. 4: Disc 1	
89	TOP	Looney Tunes Golden: Vol. 4: Disc 2	Series

Click "Move to Top"

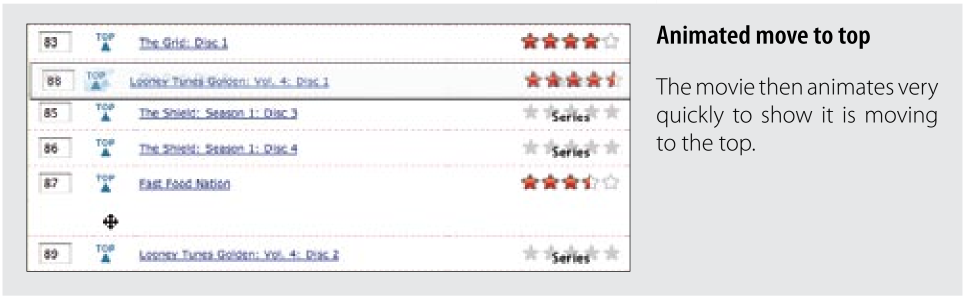
Clicking the "Move to Top" button starts the movie moving to the top.



83	TOP	The Grid: Disc 1	
84	TOP	The Grid: Disc 2	Series
85	TOP	The Shield: Season 1: Disc 3	Series
86	TOP	The Shield: Season 1: Disc 4	Series
87	TOP	Fast Food Nation	
88	TOP	Looney Tunes Golden: Vol. 4: Disc 1	
89	TOP	Looney Tunes Golden: Vol. 4: Disc 2	Series

Spring loaded

The movie does not immediately start moving up. Instead, it drops down and to the right slightly. This gives the feeling that the movie is being launched to the top.



83	TOP	The Grid: Disc 1	
88	TOP	Looney Tunes Golden: Vol. 4: Disc 1	
85	TOP	The Shield: Season 1: Disc 3	Series
86	TOP	The Shield: Season 1: Disc 4	Series
87	TOP	Fast Food Nation	
89	TOP	Looney Tunes Golden: Vol. 4: Disc 2	Series

Animated move to top

The movie then animates very quickly to show it is moving to the top.

Figure 2-21. *When a movie is moved to the top with the "Move to Top" button, the movie jerks down slightly, then springs to the top*

The combination of the downward jerk and then the quick animation to the top gives a subtle clue that the object is draggable. This is also an interesting moment to advertise drag and drop. After the move to top completes, a simple tip could appear to invite users to drag and drop. The tip should probably be shown only once, or there should be a way to turn it off. Providing an invitation within a familiar idiom is a good way to lead users to the new idiom.

Tip

If drag and drop is a secondary way to perform a task, use the completion of the familiar task as an opportunity invite the user to drag and drop the next time.

Drag lens

Drag and drop works well when a list is short or the items are all visible on the page. But when the list is long, drag and drop becomes painful. Providing alternative ways to rearrange is one way to get around this issue. Another is to provide a *drag lens* while dragging.

A drag lens provides a view into a different part of the list that can serve as a shortcut target. It could be a fixed area that is always visible, or it could be a miniature view of the list that provides more rows for targeting. The lens will be made visible only during dragging. A good example of this is dragging the insertion bar while editing text on the iPhone (Figure 2-22).

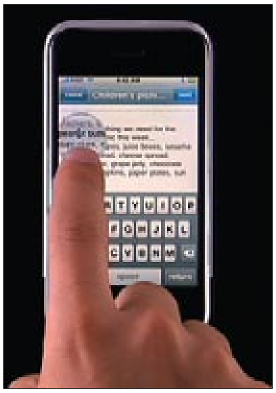


Figure 2-22. *The iPhone provides a drag magnifier lens that makes it easier to position the cursor*

Best Practices for Drag and Drop List

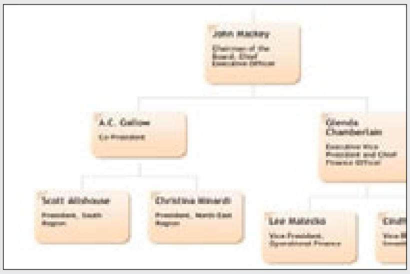
Here are some best practices to keep in mind:

- If possible, drag the items in a list in real time using the placeholder target approach.
- Use the mouse position for drag target positioning.
- If the goal is speed of dragging or if dragged items are large, consider using the insertion target approach, as rendering an insertion bar is inexpensive compared to dynamically rearranging the list.
- Since drag and drop in lists is not easily discoverable, consider providing an alternate way to rearrange the list.
- When the user rearranges the list with an alternate method, use that moment for a one-time advertisement for drag and drop.

Drag and Drop Object

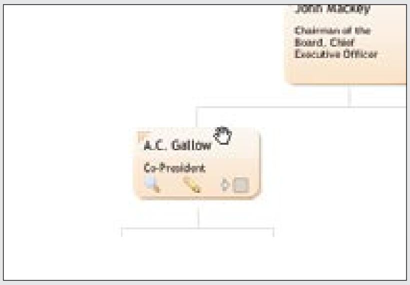
Another common use for drag and drop is to change relationships between objects. This is appropriate when the relationships can be represented visually. Drag and drop as a means of visually manipulating relationships is a powerful tool.

Cogmap is a wiki for organizational charts. **Drag and Drop Object** is used to rearrange members of the organization (Figure 2-23).


John Mackey
Chairman of the Board, Chief Executive Officer
A.C. Gallow
Co-President
Glenda Chamberlain
Executive Vice President and Chief Finance Officer
Scott Allshouse
President, South Region
Christina Minardi
President, North East Region
Lee Matecko
Vice President, Operational Finance

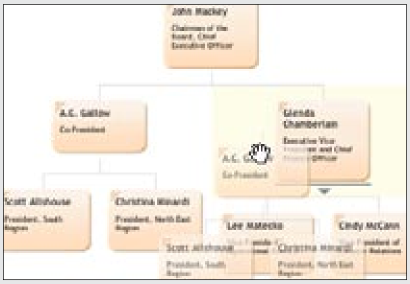
Normal display state

An organizational chart visually represents relationships.



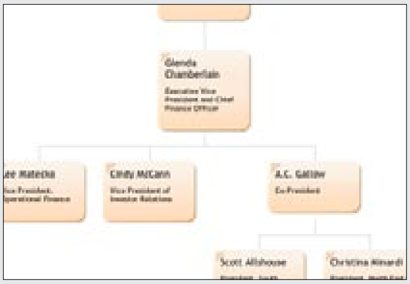
Invitation to drag

When the mouse hovers over a member of the organization, the cursor changes to show draggability. In addition, the texture in the top-left corner changes to represent a dimpled surface. This hints at draggability.



Dragging

An insertion bar is used to indicate where the member will be inserted when dropped.


Glenda Chamberlain
Executive Vice President and Chief Finance Officer
Lee Matecko
Vice President, Operational Finance
Cindy McCann
Vice President of Investor Relations
A.C. Gallow
Co-President
Scott Allshouse
Christina Minardi

Dropped

When the dragged member is dropped, the chart is rearranged to accommodate the new location.

Figure 2-23. *Cogmap allows organizational charts to be rearranged on the fly with drag and drop*

Considerations

When object relationships can be clearly represented visually, drag and drop is a natural choice to make these type of changes. Cogmap uses the target insertion approach. This allows the dragging to be nondistracting, since the chart does not have to be disturbed during targeting.

Drag feedback: Highlighting

Bubbl.us, an online mind-mapping tool, simply highlights the node that will be the new parent (Figure 2-24).

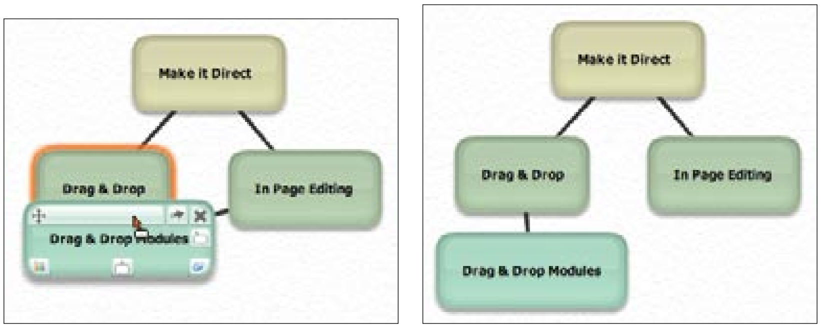


Figure 2-24. *Bubbl.us provides a visual indication of which node the dropped node will attach itself to*

In both cases, immediate preview is avoided since it is difficult to render the relationships in real time without becoming unnecessarily distracting.

Looking outside the world of the Web, the desktop application Mind Manager also uses highlighting to indicate the parent in which insertion will occur. In addition, it provides insertion targeting to give a preview of where the employee will be positioned once dropped (Figure 2-25).

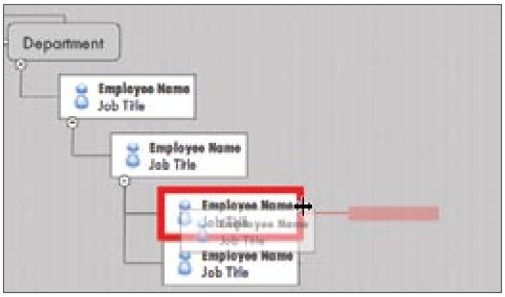
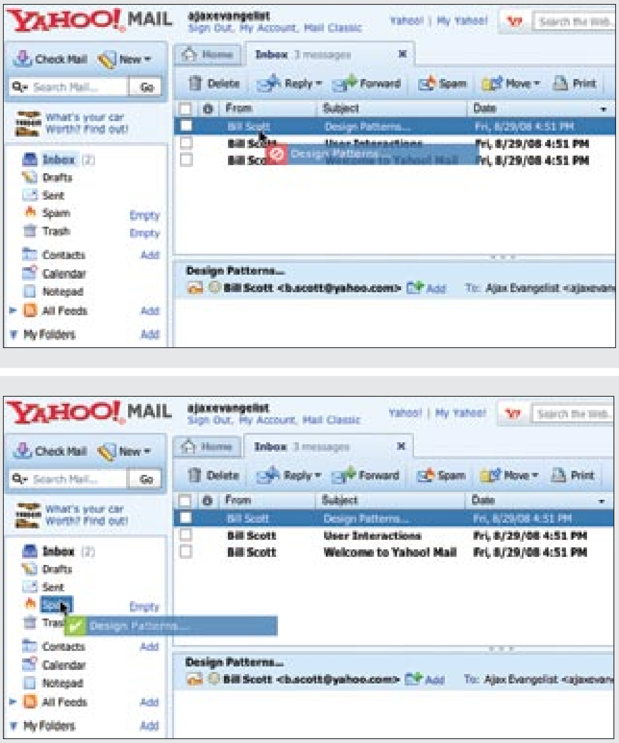


Figure 2-25. *Mind Manager is a desktop tool that uses a combination of insertion targeting plus a clear preview of the drop*

Drag feedback: Dragged object versus drop target

As we mentioned at the beginning of this chapter, one of the first serious uses for drag and drop was in the Oddpost web mail application. Oddpost was eventually acquired by Yahoo! and is now the Yahoo! Mail application.

Yahoo! Mail uses drag and drop objects for organizing email messages into folders (Figure 2-26).



From	Subject	Date
Bill Scott	Design Patterns...	Fri, 8/29/08 4:51 PM
Bill Scott	User Interactions	Fri, 8/29/08 4:51 PM
Bill Scott	Welcome to Yahoo! Mail	Fri, 8/29/08 4:51 PM

Design Patterns...
Bill Scott <b.scott@yahoo.com> Add To: Ajax Evangelist <ajaxevan

Drag initiated

When a message drag is initiated, a snippet of the message is shown, along with an icon denoting whether a drop can be made.

Valid drop target

When the dragged message may be dropped, the icon portion of the dragged object changes from a red invalid sign to a green checkmark.

Figure 2-26. *Yahoo! Mail allows messages to be dragged to folders*

Instead of signaling that a drop is valid or invalid by changing the visual appearance of the area dragged over, Yahoo! Mail shows validity through the dragged object. When a drop will be invalid (Figure 2-27, left):

- The dragged object's icon becomes a red invalid sign.
- If over an invalid folder, the folder is highlighted as well.

When a drop will be valid (Figure 2-27, right):

- The dragged object's icon changes to a green checkmark.
- The drop target highlights.

Another approach is to signal *both* validity and location in the drop target itself. In this case you would highlight the valid drop target when it is dragged over and *not* highlight the drop target if it is invalid. In Yahoo! Mail's interaction, the signaling of validity and where it can be dropped are kept separate. This allows a drag to indicate that a target is a drop target, just not valid for the current object being dragged.*

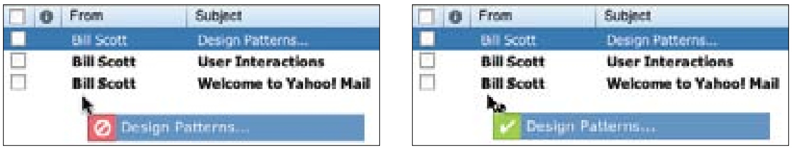


Figure 2-27. *Yahoo! Mail mistakenly shows a valid indicator instead of an invalid indicator for a message when it is dragged back over the inbox*

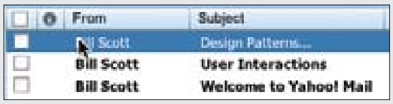
One odd situation occurs when you first start dragging a message and then later drag it back into the inbox area (Figure 2-27). At first it shows the inbox as an invalid drop area. Then it shows it as a valid drop area. Recall in our discussion on the various interesting events that initially dragging over your "home area" and then later dragging back into it are all events that should be considered during drag and drop. Here the interface needs to display the same indicator in both cases.

Tip

Feedback during dragging is key to providing a clear **Drag and Drop Object** interaction.

Drag feedback: Drag positioning

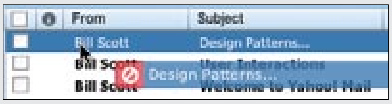
Another slightly troublesome approach is positioning the dragged object some distance away from the mouse (Figure 2-28). The reason the object is positioned in this manner is to avoid obscuring dragged-over folders. While this may alleviate that problem, it introduces a second problem: when you initiate the drag, the dragged message jumps into the offset position. Instead of conveying that the first message in the list is being dragged, it feels like the second message in the list is being dragged (Figure 2-28, bottom).



Before drag initiated

The user is about to drag the first message in the list.

* For example, while a contact may be dragged into the Contacts folder, a message may not. In either situation, the Contacts folder will highlight. However, the dragged contact will show a green checkmark, while the dragged message will show a red invalid sign.



Offset drop object

The dragged object appears below the midpoint of the second message after drag is initiated. Targeting is confusing—is it the mouse or the object that determines targeting?

Figure 2-28. *By offsetting the drag object a large distance from the cursor, the message feels disjointed from the actual object being dragged; in fact, it looks like it is closer to the second message in the list instead of the first message actually being dragged*

Drag feedback: Drag start

In Yahoo! Mail, message dragging is initiated when the mouse is dragged about four or five pixels (Figure 2-29).

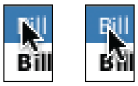


Figure 2-29. *Yahoo! Mail requires the user to drag four or five pixels to initiate a drag (notice the cursor is at the top of the "B" and has to be dragged 2/3 of the way down to start the drag); this gives the impression that the message is stuck and not easy to drag. Reducing this value will make messages feel easier to drag*

A good rule of thumb on drag initiation comes from the Apple Human Interface Guidelines:

> *Your application should provide drag feedback as soon as the user drags an item at least three pixels. If a user holds the mouse button down on an object or selected text, it should become draggable immediately and stay draggable as long as the mouse remains down.*[*]

It might seem like a small nit, but there is quite a difference between starting a drag after three pixels of movement versus four or five pixels. The larger value makes the object feel hard to pull out of its slot to start dragging. On the flip side, starting a drag with too small a value can cause drag to initiate accidentally, usually resulting in the interface feeling too finicky.

Tip

Start a drag when the object is dragged three pixels or the mouse is held down for half a second.

[*] See *http://tinyurl.com/5aqd4k* for the Apple Human Interface Guideline on drag feedback.

The only part of the Apple guideline that could be quibbled with is whether to start drag mode immediately on mouse down or wait about a half-second to start. Why not initiate the drag immediately? Certain devices, like pen input, are not as precise as mouse input. If you allow an object to be dragged and that object has other controls (like hyperlinks), you will want to allow the user to start a drag even if he clicks down over some element within the object (like a hyperlink). You will also want to allow him to just click the hyperlink and not have a drag accidentally initiate. Moving into drag mode immediately will preclude the ability to disambiguate between a click on an item within the object versus a drag start on the object itself.

Best Practices for Drag and Drop Object

Here are some best practices to keep in mind:

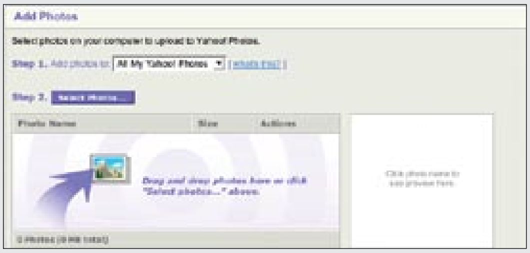
- If objects are represented in a complex visual relationship, use insertion targeting to indicate drop location (minimizes disturbing the page during drag).
- For parent/child relationships, highlight the parent as well to indicate drop location.
- If possible, reveal drag affordances on mouse hover to indicate draggability.
- Initiate drag when the mouse is dragged three pixels or if the mouse is held down for at least half a second.
- Position dragged objects directly in sync with the cursor. Offsetting will make the drag feel disjointed.
- When hovering over a draggable object, change the cursor to indicate draggability.

Drag and Drop Action

Drag and drop is also useful for invoking an action or actions on a dropped object. The **Drag and Drop Action** is a common pattern. Its most familiar example is dropping an item in the trash to perform the delete action.

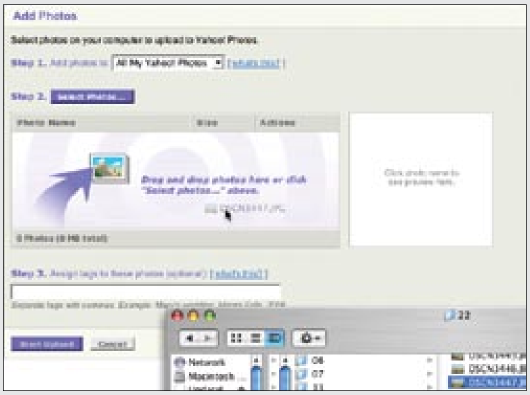
Normally uploading files to a web application includes pressing the upload button and browsing for a photo. This process is repeated for each photo.

When Yahoo! Photos was relaunched in 2006, it included a drag and drop upload feature. It allowed the user to drag photos directly into the upload page. The drop signified the upload action (Figure 2-30).


Add Photos
Select photos on your computer to upload to Yahoo! Photos.
Step 1. Add photos to: All My Yahoo! Photos [what's this?]
Step 2. Select Photos...
Photo Name
Size
Actions
Drag and drop photos here or click "Select photos..." above.
Click photo name to see preview here.
0 Photos (0 MB total)

Normal display state

"Add Photos" allows either browsing for photos or simply dragging and dropping them into the target zone below.


Add Photos
Select photos on your computer to upload to Yahoo! Photos.
Step 1. Add photos to: All My Yahoo! Photos [what's this?]
Step 2. Select Photos...
Photo Name
Size
Actions
Drag and drop photos here or click "Select photos..." above.
Click photo name to see preview here.
0 Photos (0 MB total)
Step 3. Assign tags to these photos (optional) [what's this?]
Start Upload
Cancel
Network
Macintosh
06
07
22
DSCN3448.JPG

Invitation to drag

The invitation is clear. By using the drop target area as an advertisement for the drag feature, the process is discoverable (as well as natural).

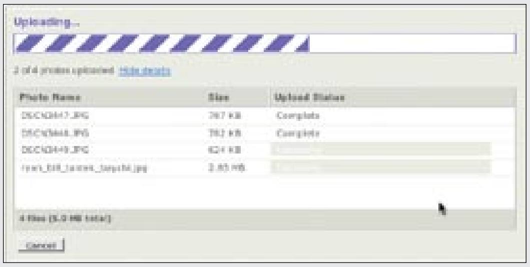


Photo Name	Size	Upload Status
DSCN3447.JPG	767 KB	Complete
DSCN3448.JPG	782 KB	Complete
DSCN3449.JPG	624 KB	
[illegible].jpg	2.85 MB	

Dropped

Photos dropped are collected into an upload area. Pressing "Start Upload" starts the uploading process.

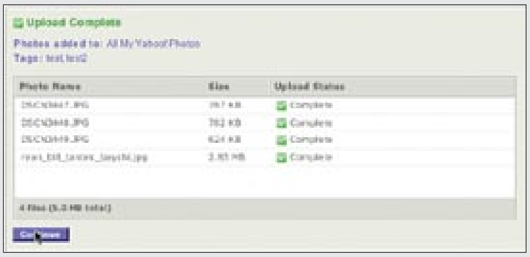


Photo Name	Size	Upload Status
DSCN3447.JPG	767 KB	✓ Complete
DSCN3448.JPG	782 KB	✓ Complete
DSCN3449.JPG	624 KB	✓ Complete
[illegible].jpg	2.85 MB	✓ Complete

Completed

All items are marked Complete when finished.

Figure 2-30. *Yahoo! Photos provided a way to upload files directly to the site via drag and drop from the user's filesystem into the web page*

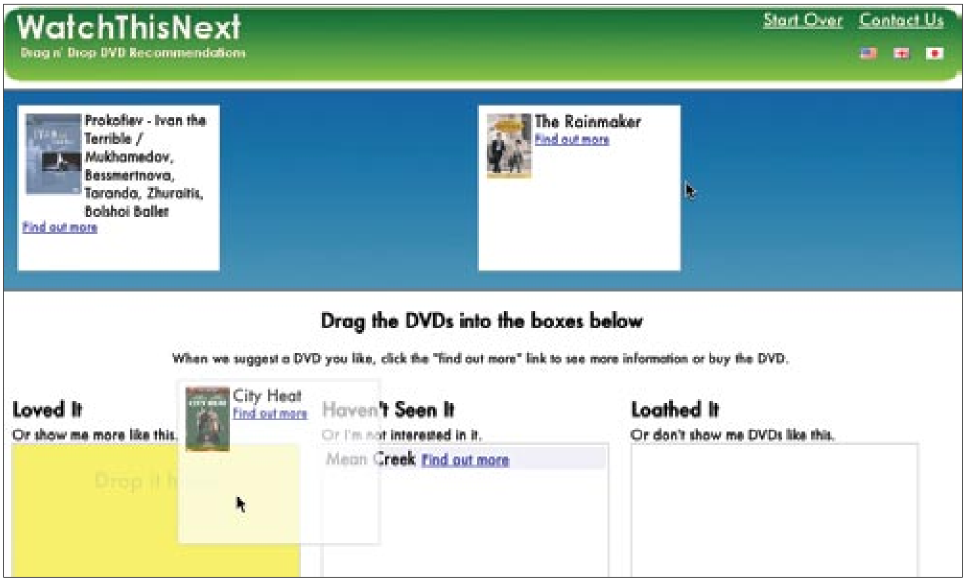
Considerations

This is not a trivial implementation. But it does clearly illustrate the benefit of drag and drop for operating on a set of files. The traditional model requires each photo to be selected individually for upload. Drag and drop frees you to use whatever browsing method is available on your system and then drop those photos for upload.

Anti-pattern: Artificial Visual Construct

Unfortunately, drag and drop can sometimes drive the design of an interface instead of being an extension of a natural interface. These interactions are almost always doomed, as they are the tail wagging the proverbial dog. Rating movies, books, and music is a common feature found on many sites. But what happens if you try to use drag and drop to rate movies?

In Figure 2-31 you can rate movies by dragging them into three buckets: "Loved It", "Haven't Seen It", or "Loathed It".


WatchThisNext
Drag n' Drop DVD Recommendations
Start Over
Contact Us
Prokofiev - Ivan the Terrible / Mukhamedov, Bessmertnova, Taranda, Zhuraitis, Bolshoi Ballet
Find out more
The Rainmaker
Find out more
City Heat
Find out more
Loved It
Or show me more like this.
Drop it here
Haven't Seen It
Or I'm not interested in it.
Mean Creek Find out more
Loathed It
Or don't show me DVDs like this.

Drag the DVDs into the boxes below

When we suggest a DVD you like, click the "find out more" link to see more information or buy the DVD.

Figure 2-31. *Drag and drop recommendations: the hard way to do ratings*

While this certainly would work, it is wrong for several reasons:

Non-obvious
: Requires some additional instructions to "Drag the DVDs into the boxes below" in order for the user to know how to rate the movies.

Too much effort

Requires too much user effort for a simple task. The user needs to employ mouse gymnastics to simply rate a movie. Drag and drop involves these discrete steps: target, then drag, then target, and then drop. The user has to carefully pick the movie, drag it to the right bucket, and release.

Too much space

Requires a lot of visual space on the page to support the idiom. Is it worth this amount of screen real estate?

Direct rating systems (thumbs up/down, star ratings, etc.) are a much simpler way to rate a movie than using an **Artificial Visual Construct**. A set of stars is an intuitive, compact, and simple way to rate a movie (Figure 2-32).

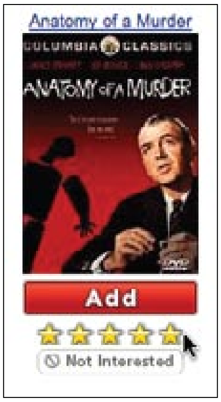


Figure 2-32. *Instead of drag and drop, Netflix uses a simple set of stars to rate a movie*

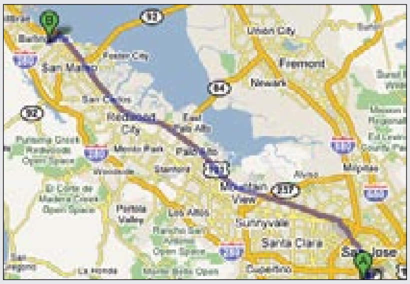
You might still be tempted to take this approach if you have a lot of objects that you want to add as favorites or set an attribute on. Don't give in. This method still falls way short since the amount of space needed for this far outweighs simpler approaches such as providing an action button for the selected objects.

Tip

Drag and drop should never be forced. Don't create an artificial visual construct to support it.

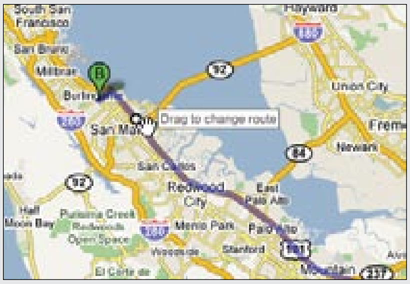
Natural Visual Construct

Another example of **Drag and Drop Action** is demonstrated in Google Maps. A route is visually represented on the map with a dark purple line. Dragging an arbitrary route point to a new location changes the route in real time (Figure 2-33).



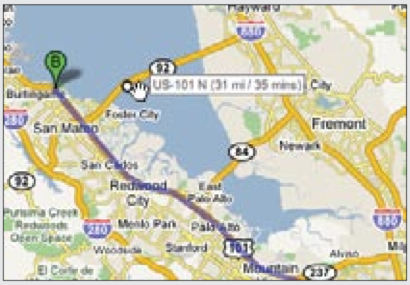
Normal display state

Route is shown in dark purple.



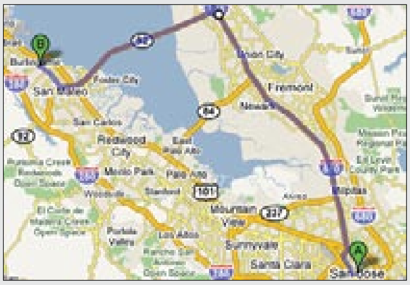
Invitation to drag

Hovering over any part of the route provides a draggable circle (route point) with a tool tip saying "Drag to change route".



Dragging

We want to stay on the east side of the bay and cross the San Mateo bridge. Dragging the route bubble back over the bridge will reroute our trip.



Dropped

The route changes as we drag. Dropping completes the rerouting action.

Figure 2-33. *Rerouting in Google Maps is as simple as drag and drop*

This is the opposite of the **Artificial Visual Construct** anti-pattern. The route is a **Natural Visual Construct**. Since anywhere along the route is draggable, there are a lot of opportunities to discover the rerouting bubble. When the route is being dragged, Google dynamically updates it. The constant feedback forms the basis of a **Live Preview** (which we will discuss in Chapter 13).

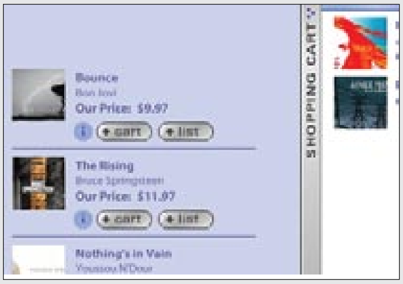
Best Practices for Drag and Drop Action

Here are some best practices to keep in mind:

- Use **Drag and Drop Actions** sparingly in web interfaces, as they are not as discoverable or expected.
- Provide alternate ways to accomplish the action. Use the **Drag and Drop Action** as a shortcut mechanism.
- Don't use drag and drop for setting simple attributes. Instead use a more direct approach to setting attributes on the object.
- Don't construct an artificial visual representation for the sole purpose of implementing drag and drop. Drag and drop should follow the natural representation of the objects in the interface.
- Provide clear invitations on hover to indicate the associated action.

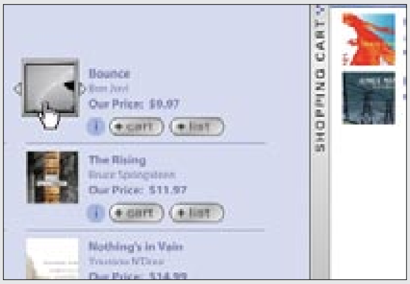
Drag and Drop Collection

A variation on dragging objects is collecting objects for purchase, bookmarking, or saving into a temporary area. This type of interaction is called **Drag and Drop Collection**. Drag and drop is a nice way to grab items of interest and save them to a list. The Laszlo shopping cart example illustrates this nicely (Figure 2-34).



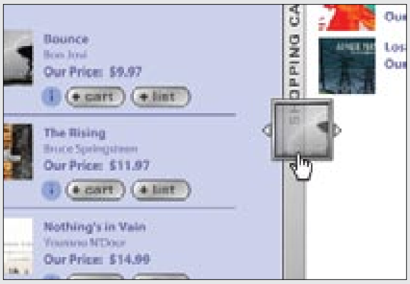
Normal display state

The shopping cart is docked on the right part of the screen.



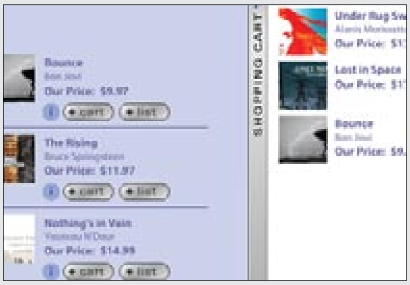
Invitation to drag

You can add to the cart with the "+ cart" button or you can drag the item to the shopping cart. If you use the button, the item flies to the cart; the cart bumps open and closed briefly to indicate that the item has been entered.



Dragging

The item gets a dragging treatment.



Dropped

The cart is populated with the new item.

Figure 2-34. *This Laszlo shopping cart demo uses both drag and drop and a button action to add items to its shopping cart*

Considerations

There are a few issues to consider in this example.

Discoverability

Drag and drop is a natural way collect items for purchase. It mimics the shopping experience in the real world. Grab an item. Drop it in your basket. This is fast and convenient once you know about the feature. However, as a general rule, you should never rely solely on drag and drop for remembering items.

Parallel, more explicit ways to do the same action should be provided. In this example, Laszlo provides an alternative to dragging items in the cart. Notice the “+ cart” button in Figure 2-34. Clicking this button adds the item to the shopping cart.

Best Practices for Drag and Drop Collection

Here are some best practices to keep in mind:

- Use as an alternate way to collect items (e.g., a shopping cart).
- When a drag gets initiated, highlight the valid drop area to hint where drop is available.
- Provide alternate cues that drag and drop into collections as available.

Teachable moment

When providing alternates to drag and drop, it is a good idea to hint that dragging is an option. In the Laszlo example, clicking the “+ cart” button causes the shopping cart tray to bump slightly open and then closed again. This points to the physicality of the cart. Using another interaction as a teachable moment to guide the user to richer interactions is a good way to solve discoverability issues.

Tip

Look for opportunities for teachable moments in the interface leading users to advanced features.

The Challenges of Drag and Drop

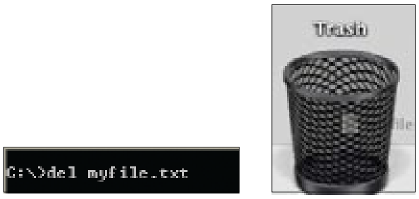
As you can see from the discussion in this chapter, Drag and Drop is complex. There are four broad areas where Drag and Drop may be employed: **Drag and Drop Module**, **Drag and Drop List**, **Drag and Drop Object**, and **Drag and Drop Action**. And in each area, there are a large number of interesting moments that may be handled in numerous ways. Being consistent in visual and interaction styles across all of these moments for all of these types of interactions is a challenge in itself. And keeping the user informed throughout the process with just the right amount of hints requires design finesse. In Chapter 10, we explore some ways to bring this finesse into Drag and Drop.

General Best Practices for Drag and Drop

- Keep page jitter to a minimum while dragging objects.
- Initiate dragging if the user presses the mouse down and moves the mouse three pixels, or if she holds the mouse down for at least half a second.
- Use drag and drop for performing direct actions as an alternate method to more direct mechanisms in the interface.
- Hint at the availability of drag and drop when using alternatives to drag and drop.
- Pay attention to all of the interesting moments during drag and drop. Remember, you must keep the user informed throughout the process.
- Use **Invitations** (discussed more in Chapters 9 and 10) to cue the user that drag and drop is available.

Chapter 3
Direct Selection

When the Macintosh was introduced, it ushered into the popular mainstream the ability to directly select objects and apply actions to them. Folders and files became first-class citizens. Instead of a command line to delete a file, you simply dragged a file to the trash-can (Figure 3-1).



```
C:\>del myfile.txt
```

Figure 3-1. *DOS command line for deleting a file versus dragging a file to the trash on the Macintosh*

Treating elements in the interface as directly selectable is a clear application of the *Make It Direct* principle. On the desktop, the most common approach is to initiate a selection by directly clicking on the object itself. We call this selection pattern **Object Selection** (Figure 3-2).

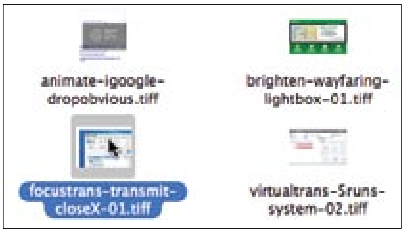

animate-igoogle-dropobvious.tiff
brighten-wayfaring-lightbox-01.tiff
focustrans-transmit-closeX-01.tiff
virtualtrans-5runs-system-02.tiff

Figure 3-2. *Files can be selected directly on the Macintosh; Object Selection is the most common pattern used in desktop applications*

In this chapter we will look at the following types of selection patterns:

Toggle Selection
: Checkbox or control-based selection.

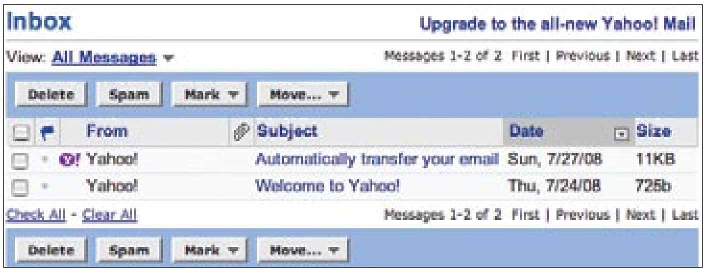
Collected Selection
: Selection that spans multiple pages.

Object Selection
: Direct object selection.

Hybrid Selection
: Combination of **Toggle Selection** and **Object Selection**.

Toggle Selection

The most common form of selection on the Web is **Toggle Selection**. Checkboxes and toggle buttons are the familiar interface for selecting elements on most web pages. An example of this can be seen in Figure 3-3 with Yahoo! Mail Classic.


Inbox
Upgrade to the all-new Yahoo! Mail
View: All Messages
Messages 1-2 of 2 First | Previous | Next | Last
Delete
Spam
Mark
Move...
Check All - Clear All
Messages 1-2 of 2 First | Previous | Next | Last
Delete
Spam
Mark
Move...

From	Subject	Date	Size
Yahoo!	Automatically transfer your email	Sun, 7/27/08	11KB
Yahoo!	Welcome to Yahoo!	Thu, 7/24/08	725b

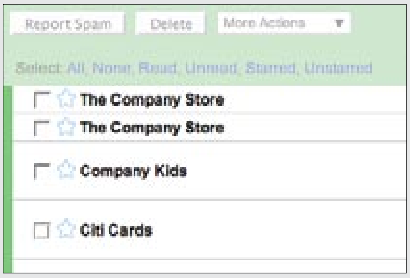
Figure 3-3. *In Yahoo! Mail Classic a mail message can be selected by clicking on the corresponding row's checkbox*

The way to select an individual mail message is through the row's checkbox. Clicking on the row itself does not select the message. We call this pattern of selection **Toggle Selection** since toggle-style controls are typically used for selecting items.

Tip

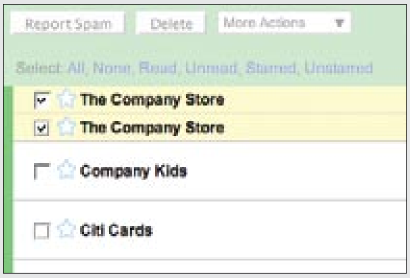
Toggle Selection is the easiest way to allow discontinuous selection.

Once items have been check-selected, actions can be performed on them. Usually these actions are performed on the selection by clicking on a separate button (e.g., the Delete button). Gmail is a good example of actions in concert with **Toggle Selection** (Figure 3-4).



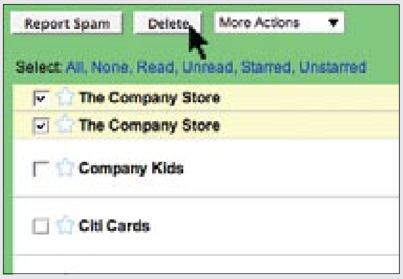
Unselected state

Each mail message has a checkbox that controls whether it is selected or not.



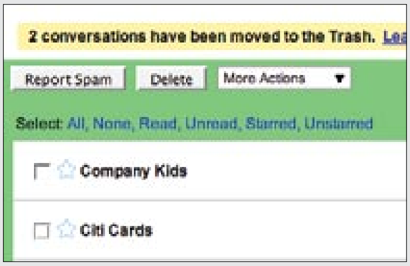
Selected items

Two mail messages have been selected. In addition to the checkbox selection, the selected items are highlighted in light yellow.



Action triggered

Delete will operate on the selected items.


2 conversations have been moved to the Trash. Lea

Action completed

The two selected email messages have been deleted.

Figure 3-4. *Gmail uses checkbox selection to operate on messages*

Considerations

Toggle Selection with checkboxes has some nice attributes:

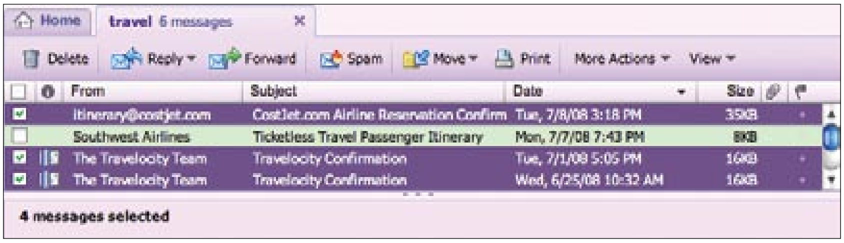
- Clear targeting, with no ambiguity about how to select the item or deselect it.
- Straightforward discontinuous selection, and no need to know about Shift or Control-key ways to extend a selection. Just click the checkboxes in any order, either in a continuous or discontinuous manner.
- Clear indication of what has been selected.

Scrolling versus paging

The previous examples were with paged lists. But what about a scrolled list? Yahoo! Mail uses a scrolled list to show all of its mail messages (Figure 3-5). While not all messages are visible at a time, the user knows that scrolling through the list retains the currently selected items. Since the user understands that all the messages not visible are still on the same continuous pane, there is no confusion about what an action will operate on—it will affect all selected items in the list. Sometimes the need for clarity of selection will drive the choice between scrolling and paging.

Tip

Toggle Selection is the normal pattern used when content is paged. Actions normally apply only to the selected items on the visible page.



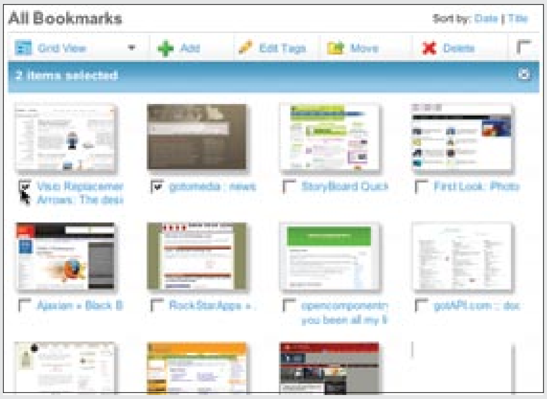
	From	Subject	Date	Size
✓	itinerary@costjet.com	CostJet.com Airline Reservation Confirm	Tue, 7/8/08 3:18 PM	35KB
☐	Southwest Airlines	Ticketless Travel Passenger Itinerary	Mon, 7/7/08 7:43 PM	8KB
✓	The Travelocity Team	Travelocity Confirmation	Tue, 7/1/08 5:05 PM	16KB
✓	The Travelocity Team	Travelocity Confirmation	Wed, 6/25/08 10:32 AM	16KB

4 messages selected

Figure 3-5. *Yahoo! Mail uses a scrolled list for its messages; selection includes what is in the visible part of the list as well as what is scrolled out of view*

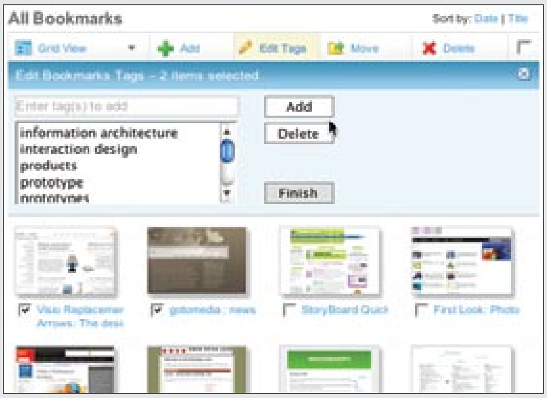
Making selection explicit

With Yahoo! Bookmarks you can manage your bookmarks by selecting bookmarked pages and then acting on them. The selection model is visually explicit (Figure 3-6).



Selected items

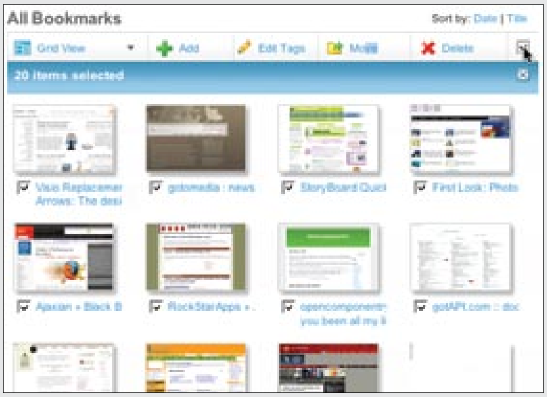
When items get selected, a status bar appears that keeps a tally of the number of items selected. The close button (x) is an alternate way to de-select the selected items.



Tools act on selection

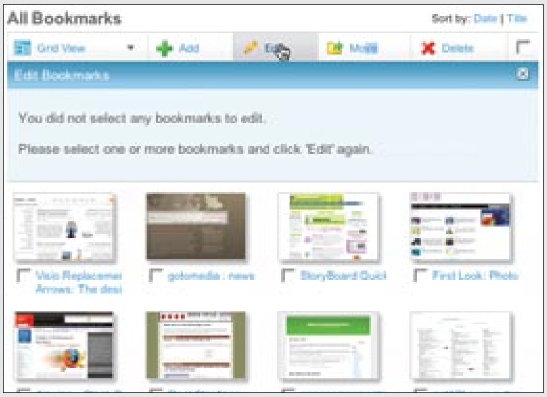
When a tool is activated ("Edit Tags") a command area slides into place.

The status area becomes the title bar for the command area.



Select all

A "select all" checkbox selects all items on the page. The selection status then shows the current number of items selected.



You did not select any bookmarks to edit.

Please select one or more bookmarks and click 'Edit' again.

No selection

If a tool is chosen and there is nothing selected, a message appears explaining why the action could not take place.

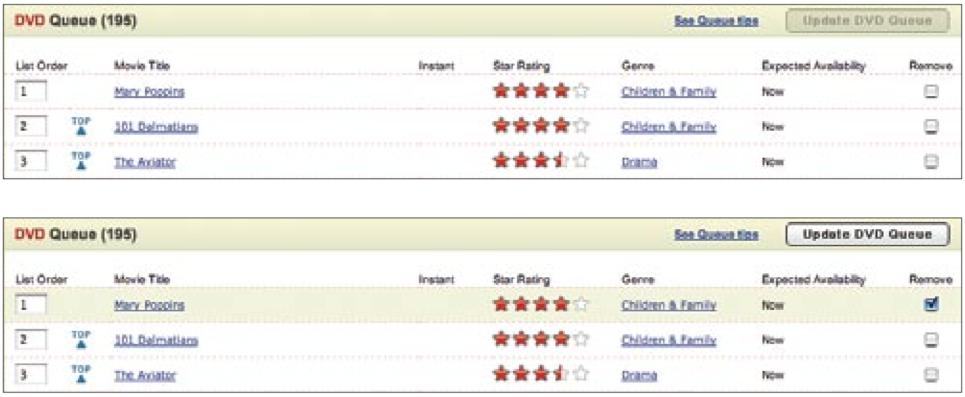
This is opposite of the normal error-prevention technique of disabling the tool when nothing is selected.

Figure 3-6. *Yahoo! Bookmarks explicitly displays the state of the selection*

The advantage of this method is that it is always clear how many items have been selected. Visualizing the underlying selection model is generally a good approach. This direct approach to selection and acting on bookmarks creates a straightforward interface.

One interesting question: what happens when nothing is selected? One approach is to disable any actions that require at least one selected item. Yahoo! Bookmarks takes a different approach. Since buttons on the Web do not follow a standard convention, you often can't rely on a color change to let you know something is not clickable. Yahoo! Bookmarks chose to make selection very explicit and make it clear when a command is invalid because nothing is selected ("No selection" in Figure 3-6). This is not normally the optimal way to handle errors. Generally, the earlier you can prevent errors, the better the user experience.

Netflix disables the "Update DVD Queue" button when nothing is selected and enables it when a movie gets selected (Figure 3-7).



List Order		Movie Title	Instant	Star Rating	Genre	Expected Availability	Remove
1		Mary Poppins		★★★★☆	Children & Family	Now	☐
2	TOP	101 Dalmatians		★★★★☆	Children & Family	Now	☐
3	TOP	The Aviator		★★★½☆	Drama	Now	☐

List Order		Movie Title	Instant	Star Rating	Genre	Expected Availability	Remove
1		Mary Poppins		★★★★☆	Children & Family	Now	☑
2	TOP	101 Dalmatians		★★★★☆	Children & Family	Now	☐
3	TOP	The Aviator		★★★½☆	Drama	Now	☐

Figure 3-7. *When nothing is selected, Netflix disables the "Update DVD Queue" button to prevent errors early*

Best Practices for Toggle Selection

Here are some best practices to keep in mind:

- Use **Toggle Selection** for selecting elements in a row.
- Use **Toggle Selection** to make it easy to select discontinuous elements.
- In a list, highlight the row in addition to the checkbox to make the selection explicit.
- When moving from page to page, actions should only operate on the items selected on that page.
- If offering a "select all" option, consider providing a way to select all elements across all pages.
- Provide clear feedback for the number of elements selected.
- If possible, disable unavailable actions when nothing is selected. If you keep the action enabled, you will need additional interface elements to signal that it can't be completed.

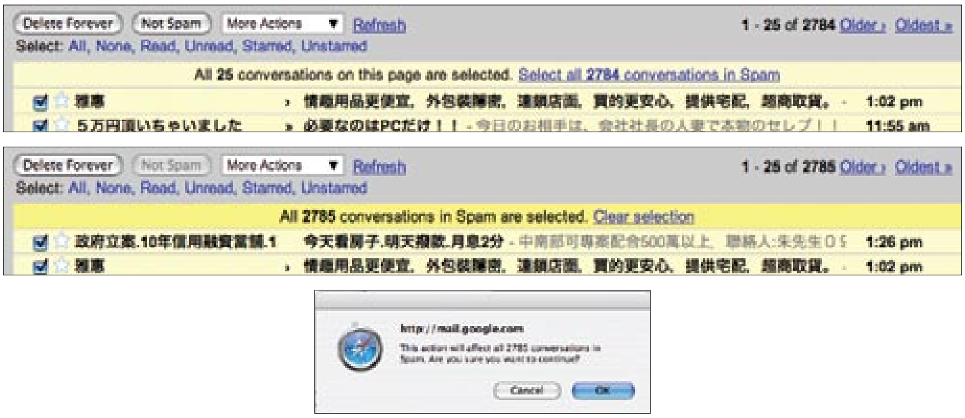
Collected Selection

Toggle Selection is great for showing a list of items on a single page. But what happens if you want to collect selected items across multiple pages? **Collected Selection** is a pattern for keeping track of selection as it spans multiple pages.

In Gmail, you can select items as you move from page to page. The selections are remembered for each page. If you select two items on page one, then move to page two and select three items, there are only three items selected. This is because actions only operate on a single page. This makes sense, as users do not normally expect selected items to be remembered across different pages.

Considerations

Gmail does provide a way to select all items across different pages. When selecting all items on a individual page (with the "All" link), a prompt appears inviting the user to "Select all 2785 conversations in Spam". Clicking that will select all items across all pages (Figure 3-8). The "Delete Forever" action will operate on all 2785 conversations, not just the 25 selected on the page.


Delete Forever
Not Spam
More Actions
Refresh
1 - 25 of 2784 Older › Oldest »
Select: All, None, Read, Unread, Starred, Unstarred
Delete Forever
Not Spam
More Actions
Refresh
1 - 25 of 2785 Older › Oldest »
Select: All, None, Read, Unread, Starred, Unstarred
Cancel
OK

All 25 conversations on this page are selected. Select all 2784 conversations in Spam

雅惠	› 情趣用品更便宜、外包裝隱密、連鎖店面、買的更安心、提供宅配、超商取貨。 -	1:02 pm
５万円頂いちゃいました	» 必要なのはPCだけ！！ - 今日のお相手は、会社社長の人妻で本物のセレブ！！	11:55 am

All 2785 conversations in Spam are selected. Clear selection

政府立案.10年信用融資當舖.1	今天看房子.明天撥款.月息2分 - 中南部可專案配合500萬以上 聯絡人:朱先生０５	1:26 pm
雅惠	› 情趣用品更便宜、外包裝隱密、連鎖店面、買的更安心、提供宅配、超商取貨。 -	1:02 pm

http://mail.google.com

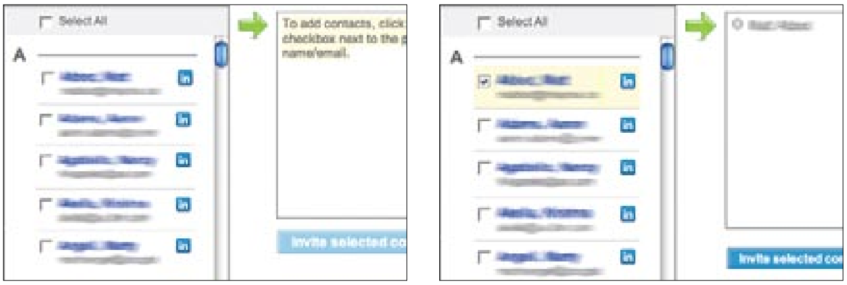
This action will affect all 2785 conversations in Spam. Are you sure you want to continue?

Figure 3-8. *Gmail provides a way to select all items across all pages, allowing the user to delete all items in a folder without having to delete all items on each page individually*

Keeping the selection visible

The real challenge for multi-page selection is finding a way to show selections gathered across multiple pages. You need a way to collect and show the selection as it is being created. Here is one way that **Collected Selection** comes into play.

LinkedIn uses **Collected Selection** to add potential contacts to an invite list (Figure 3-9).



To add contacts, click checkbox next to the name/email.

Figure 3-9. *LinkedIn provides a holding place for saving selections across multiple pages*

The list of potential invitees is shown in a paginated list on the lefthand side. Clicking the checkbox adds them to the invite list. The invite list becomes the place where selected contacts across multiple pages are remembered.

Any name in the invite list can be removed by clicking the "X" button beside it. Once the complete list of invitees is selected, clicking the "Invite selected contacts" sends each selected contact a LinkedIn invitation.

Collected Selection and actions

When Yahoo! Photos was working its way through an early design of its Photo Gallery (see Figure 3-13, later in this chapter), the plan was to show all photos in a single, continuous scrolling page (we discuss virtual scrolling in Chapter 7). In a long virtual list, the selection model is simple. Photos are shown in a single page and selection is easily understood in the context of this single page.

However, due to performance issues, the design was changed. Instead of a virtual page, photos had to be chunked into pages. In order to support **Collected Selection**, Yahoo! Photos introduced the concept of a "tray" into the interface (Figure 3-10). On any page, photos can be dragged into the tray. The tray keeps its contents as the user moves from page to page. So, adding a photo from page one and three more from page four would yield four items in the tray. As a nice touch, the tray would make itself visible (by sliding into view) even when the user was scrolled down below the fold.

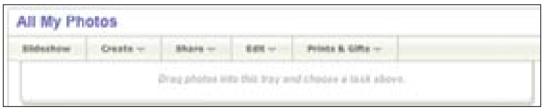


Figure 3-10. *Yahoo! Photos used a "tray" to implement a form of Collected Selection; the confusing aspect was which actions in the menu operated on the tray versus the photos selected on the page*

There was a problem with the design, however. In the menu system it was hard to discern whether the user meant to operate on the selection (photos on the page could be selected through an **Object Selection** model) or on the collected items in the tray. To resolve this ambiguity, the drop-down menus contained two identical sets of commands. The first group of commands in the menu operated on the collected items in the tray. The second set of commands operated on the selected objects. Needless to say, this was confusing since it required the user to be fully aware of these two selection models when initiating a command.

One way to remove this ambiguity would have been to have a single set of commands that operated on either the tray or the photos—depending on which had the focus. This would require a way to select the tray and a way to deselect it (by clicking outside the tray). A possible approach would be to slightly dim the photo gallery when the tray is selected (causing it to clearly have the focus), and do the opposite when the tray is not the focus.

Best Practices for Collected Selection

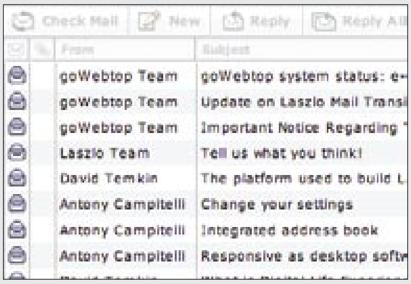
Here are some best practices to keep in mind:

- If you allow selection across page boundaries, accumulate the selected items (from each page) into a separate area. This makes the selection explicit even when moving from page to page.
- Use **Collected Selection** to blend **Toggle Selection** and **Object Selection** in the same interface.
- Watch out for ambiguity between items selected with **Collected Selection** and any items or objects that can be normally selected on the page.

Object Selection

As we mentioned earlier, **Toggle Selection** is the most common type of selection on the Web. The other type of selection, **Object Selection**, is when selection is made directly on objects within the interface.

Sometimes using a checkbox does not fit in with the style of interaction desired. Laszlo's WebTop mail allows the user to select messages by clicking anywhere in the row. The result is that the whole row gets highlighted to indicate selection (Figure 3-11).

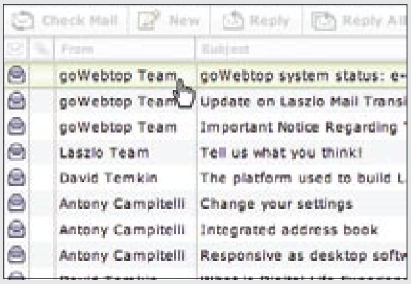


Check Mail | New | Reply | Reply Al

From	Subject
goWebtop Team	goWebtop system status: e-
goWebtop Team	Update on Laszlo Mail Transi
goWebtop Team	Important Notice Regarding
Laszlo Team	Tell us what you think!
David Temkin	The platform used to build L
Antony Campitelli	Change your settings
Antony Campitelli	Integrated address book
Antony Campitelli	Responsive as desktop softw

Nothing selected

Normal view when nothing is selected and the mouse is not over a message.

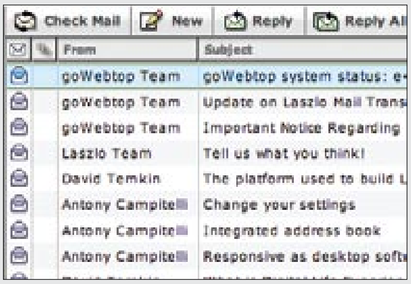


Check Mail | New | Reply | Reply Al

From	Subject
goWebtop Team	goWebtop system status: e-
goWebtop Team	Update on Laszlo Mail Transi
goWebtop Team	Important Notice Regarding
Laszlo Team	Tell us what you think!
David Temkin	The platform used to build L
Antony Campitelli	Change your settings
Antony Campitelli	Integrated address book
Antony Campitelli	Responsive as desktop softw

Hovered state

When the mouse hovers over a row, the row is subtly highlighted to indicate focus and what will be selected if the user clicks.



From	Subject
goWebtop Team	goWebtop system status: e-
goWebtop Team	Update on Laszlo Mail Trans
goWebtop Team	Important Notice Regarding
Laszlo Team	Tell us what you think!
David Temkin	The platform used to build L
Antony Campitelli	Change your settings
Antony Campitelli	Integrated address book
Antony Campitelli	Responsive as desktop softw

Selected state

When the user clicks on a message, the whole row gets selected.

Figure 3-11. *Laszlo WebTop Mail uses highlighting to indicate row selection*

Considerations

Desktop applications tend to use **Object Selection**. It is also natural that web-based mail applications that mimic desktop interactions employ this same style of selection. Instead of showing a control (like a checkbox), the object itself can be selected and acted on directly.

Object Selection can be extended by holding down the Shift key while clicking on a different item. The Command key (Macintosh) or Control key (Windows) can be used to individually add items in a discontinuous manner. The downside to this approach is that it is not obvious to use the modifier keys for extending the selection. **Toggle Selection**'s use of toggle buttons makes the selection extension model completely obvious.

Flickr is a simple example of the keyboard being used to extend the selection in a web application. In the Organizr tool, multiple photos can be selected by using modifier keys to extend the selection (Figure 3-12).

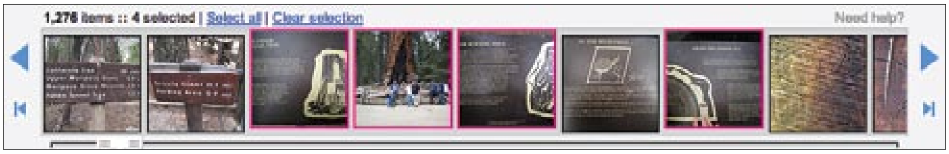


Figure 3-12. *Flickr allows for discontinuous selection by using the Command/Control key to extend selection*

Desktop-style selection

For now **Object Selection** is not as common on the Web. Given that most sites have been content-oriented, there have been few objects to select. Also, with the Web's simple event model, **Object Selection** was not easy to implement. In typical web pages, keyboard events have rarely made sense since they are also shared with the browser. However, all of this is changing as the capabilities of web technologies continue to improve.

Most desktop **Object Selection** interactions include ways to use the mouse to drag-select objects. Yahoo! Photos introduced this same type of object selection to its photo gallery (Figure 3-13). Individually clicking on a photo selects it. Using the Shift key and clicking also extends the selection. In addition, using the Control key and clicking discontinuously selects photos. And like most desktop applications, you can drag a selection box around a group of items to add them to the selected set (in this case, photos).

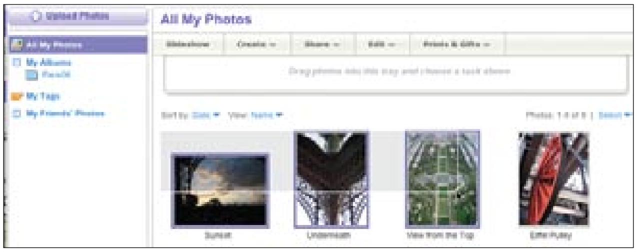


Figure 3-13. *Yahoo! Photos 3.0 created a rich drag selection mechanism for selecting photos*

Best Practices for Object Selection

Here are some best practices to keep in mind:

- Use **Object Selection** when selectable elements can be dragged.
- Use **Object Selection** when the application will simulate desktop style interactions.
- Allow standard modifier key extensions (Shift to extend selection; Ctrl for discontinuous selection).
- If possible, degrade **Object Selection** to **Toggle Selection** when browser capabilities are limited.

Hybrid Selection

Mixing **Toggle Selection** and **Object Selection** in the same interface can lead to a confusing interface. Referring back to Yahoo! Bookmarks, you'll see an odd situation arise during drag and drop (Figure 3-14).

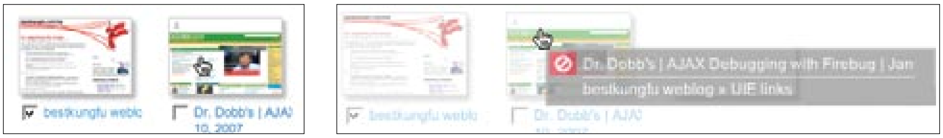

bestkungfu weblo
Dr. Dobb's | AJAX
10, 2007
Dr. Dobb's | AJAX Debugging with Firebug | Jan
bestkungfu weblog » UIE links

Figure 3-14. *In Yahoo! Bookmarks, one item is selected, but two items can be dragged by dragging on the unselected item*

Considerations

There are a few important issues to consider when using **Hybrid Selection**.

Confusing two models

In the left panel of Figure 3-14, one bookmark element is selected (notice the checkbox **Toggle Selection**). The second bookmark element ("Dr. Dobb's") is unselected (the checkbox is clear). In the right panel of Figure 3-14, clicking and dragging on the unselected bookmark element initiates a drag. The drag includes both the selected element and the unselected element. Since only one is shown as selected, this creates a confusing situation.

This occurs because three things are happening in the same space:

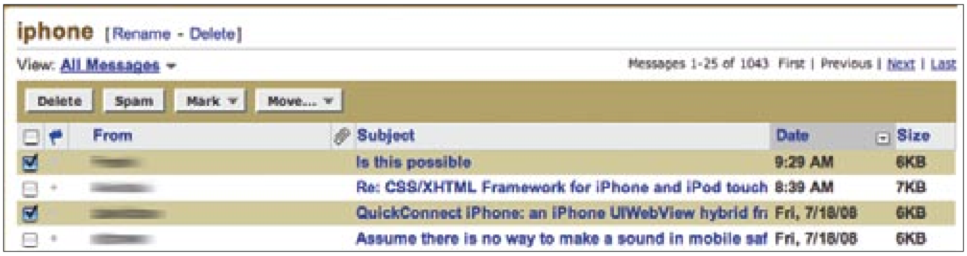
- **Toggle Selection** is used for selecting bookmarks for editing, deleting, etc.
- **Object Selection** is used for initiating a drag drop.
- Mouse click is used to open the bookmark on a separate page.

The problem is that more than one interaction idiom is applied to the same place on the same page. In this case, if you happen to try to drag, but instead click, you will be taken to a new page. And if you drag an unselected item, you now have two items selected for drag but only one shown as selected for other operations (Figure 3-14, right). This is definitely confusing. Simply selecting the item (automatically checking the box) when the drag starts would keep the selection model consistent in the interface. However, it might lead the user to expect a single click to also do the same (which it cannot since it opens the bookmark).

So, mixing the two selection models together can be problematic. However, there is a way to integrate the **Toggle Selection** and **Object Selection** and have them coexist peacefully as well as create an improved user experience.

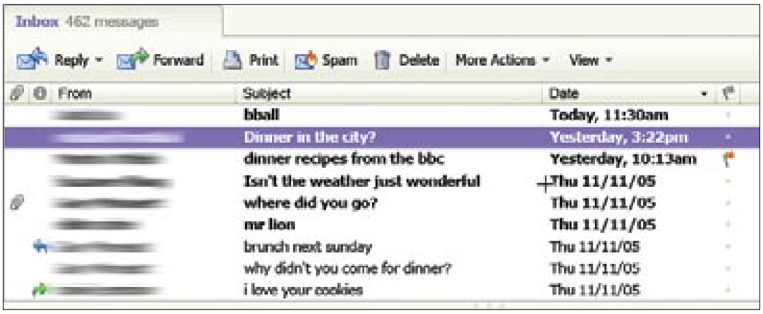
Blending two models

Yahoo! Mail originally started with the **Toggle Selection** model (Figure 3-15). When the new Yahoo! Mail Beta was released, it used **Object Selection** exclusively (Figure 3-16). But since there are advantages to both approaches, the most recent version of Yahoo! Mail incorporates both approaches in a **Hybrid Selection** (Figure 3-17).



	From	Subject	Date	Size
✓		Is this possible	9:29 AM	6KB
☐		Re: CSS/XHTML Framework for iPhone and iPod touch	8:39 AM	7KB
✓		QuickConnect iPhone: an iPhone UIWebView hybrid fr	Fri, 7/18/08	6KB
☐		Assume there is no way to make a sound in mobile saf	Fri, 7/18/08	6KB

Figure 3-15. *Yahoo! Mail Classic uses Toggle Selection; it also highlights selected rows, but rows can only be selected by clicking the message's checkbox*


Inbox 462 messages

Reply ▾ Forward Print Spam Delete More Actions ▾ View ▾

From	Subject	Date
[redacted]	bball	Today, 11:30am
[redacted]	Dinner in the city?	Yesterday, 3:22pm
[redacted]	dinner recipes from the bbc	Yesterday, 10:13am
[redacted]	Isn't the weather just wonderful	Thu 11/11/05
[redacted]	where did you go?	Thu 11/11/05
[redacted]	mr lion	Thu 11/11/05
[redacted]	brunch next sunday	Thu 11/11/05
[redacted]	why didn't you come for dinner?	Thu 11/11/05
[redacted]	i love your cookies	Thu 11/11/05

Figure 3-16. *Yahoo! Mail Beta launched with Object Selection: no checkboxes were provided, and discontiguous selection could only be done by using keyboard modifiers*

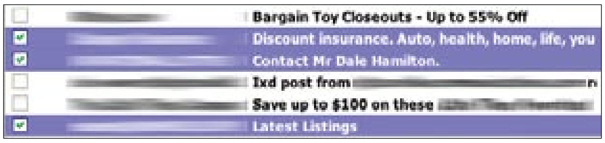

☐	[redacted]	Bargain Toy Closeouts - Up to 55% Off
✓	[redacted]	Discount insurance. Auto, health, home, life, you
✓	[redacted]	Contact Mr Dale Hamilton.
☐	[redacted]	Ixd post from [redacted]
☐	[redacted]	Save up to $100 on these [redacted]
✓	[redacted]	Latest Listings

Figure 3-17. *Yahoo! Mail now uses a hybrid approach: it incorporates both the Toggle Selection and the Object Selection patterns (messages can be selected with checkboxes or by clicking on the row); Toggle Selection selects the message without loading the message in the viewing pane*

Hybrid Selection brings with it the best of both worlds. You can use the checkbox selection model as well as normal row selection. You get the benefit of explicit selection and simplified multiple selection that **Toggle Selection** brings. And you get the benefit of interacting with the message itself and direct object highlighting.

Tip

Combining **Toggle Selection** and **Object Selection** is a nice way to bridge a common web idiom with a common desktop idiom.

There is an additional meaning applied to **Toggle Selection** versus **Object Selection**. Clicking on a row with the checkbox has the benefit of selecting the message without loading its contents in the message pane (think spam!). Clicking on a message itself will load the contents in the message pane.

Best Practices for Hybrid Selection

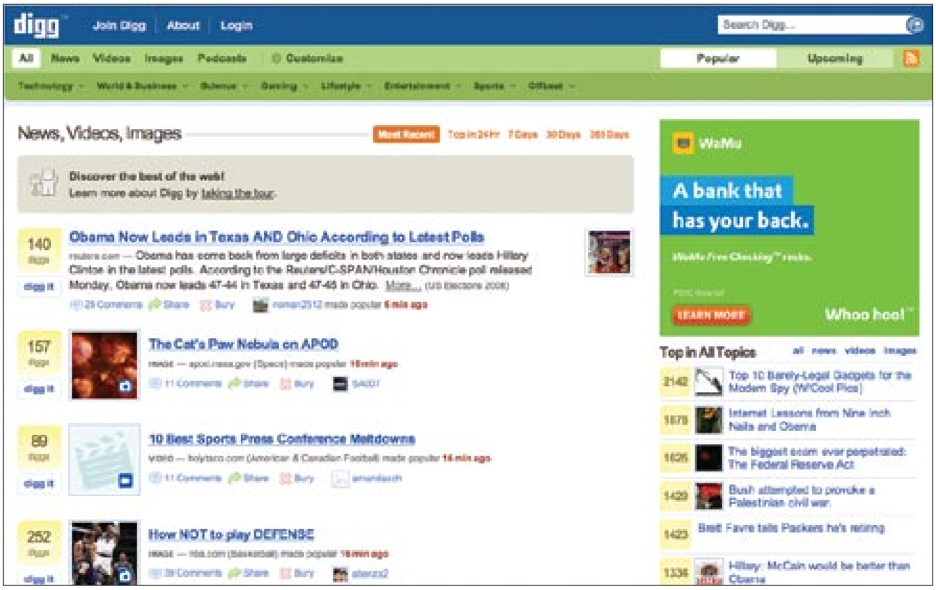
Here are some best practices to keep in mind:

- Use checkbox selection to select an object without opening it.
- Use object selection to select and open an object.

Principle Two

Keep It Lightweight

Digg is a popular news site where the community votes on its favorite stories. If you see a story on Digg that you like you can vote your approval with a simple click action on the “digg it” button (Figure P2-1).


digg
Join Digg | About | Login
Search Digg...
All News Videos Images Podcasts Customize
Popular Upcoming
Technology World & Business Science Gaming Lifestyle Entertainment Sports Offbeat
News, Videos, Images
Most Recent Top in 24 Hr 7 Days 30 Days 365 Days
WaMu
A bank that has your back.
WaMu Free Checking™ rocks.
FDIC Insured
LEARN MORE
Whoo hoo!™
Top in All Topics
all news videos images

Discover the best of the web!
Learn more about Digg by taking the tour.

140 diggs digg it — **Obama Now Leads in Texas AND Ohio According to Latest Polls**
reuters.com — Obama has come back from large deficits in both states and now leads Hillary Clinton in the latest polls. According to the Reuters/C-SPAN/Houston Chronicle poll released Monday, Obama now leads 47-44 in Texas and 47-45 in Ohio. More... (US Elections 2008)
25 Comments Share Bury roman2512 made popular 6 min ago

157 diggs digg it — **The Cat's Paw Nebula on APOD**
IMAGE — apod.nasa.gov (Space) made popular 15 min ago
11 Comments Share Bury SAD07

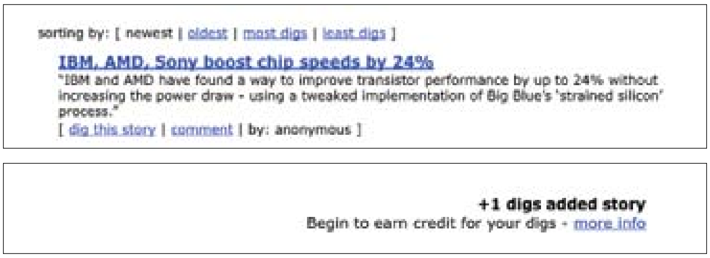
89 diggs digg it — **10 Best Sports Press Conference Meltdowns**
VIDEO — holytaco.com (American & Canadian Football) made popular 16 min ago
11 Comments Share Bury amandasch

252 diggs digg it — **How NOT to play DEFENSE**
IMAGE — nba.com (Basketball) made popular 16 min ago
39 Comments Share Bury aberzo2

2142	Top 10 Barely-Legal Gadgets for the Modern Spy (W/Cool Pics)
1879	Internet Lessons from Nine Inch Nails and Obama
1626	The biggest scam ever perpetrated: The Federal Reserve Act
1429	Bush attempted to provoke a Palestinian civil war.
1423	Brett Favre tells Packers he's retiring
1336	Hillary: McCain would be better than Obama

Figure P2-1. *Digg makes it easy to vote to digg articles with the “digg it” button next to each article*

It wasn’t always this simple. Kevin Rose founded Digg to democratize the finding of tech articles. In the earliest versions of Digg, the user had to complete a two-step process to digg a story (Figure P2-2).



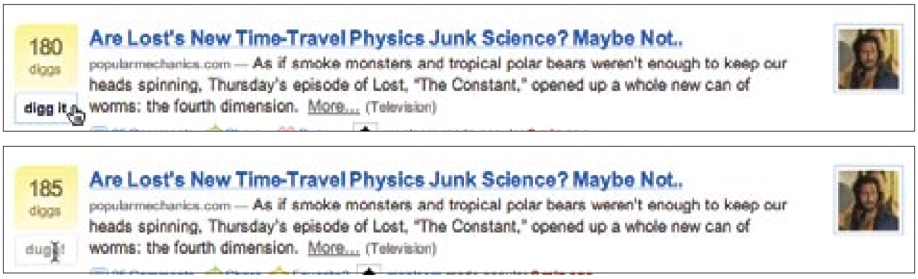
sorting by: [newest | oldest | most digs | least digs]

IBM, AMD, Sony boost chip speeds by 24%
"IBM and AMD have found a way to improve transistor performance by up to 24% without increasing the power draw - using a tweaked implementation of Big Blue's 'strained silicon' process."
[dig this story | comment | by: anonymous]

+1 digs added story
Begin to earn credit for your digs - more info

Figure P2-2. *The first version of Digg required a two-page process to “digg” a story*

In the current version of Digg it’s much simpler—just click the “digg it” button, and the vote is immediately recorded (Figure P2-3).



Are Lost's New Time-Travel Physics Junk Science? Maybe Not..
popularmechanics.com — As if smoke monsters and tropical polar bears weren't enough to keep our heads spinning, Thursday's episode of Lost, "The Constant," opened up a whole new can of worms: the fourth dimension. More... (Television)

Are Lost's New Time-Travel Physics Junk Science? Maybe Not..
popularmechanics.com — As if smoke monsters and tropical polar bears weren't enough to keep our heads spinning, Thursday's episode of Lost, "The Constant," opened up a whole new can of worms: the fourth dimension. More... (Television)

Figure P2-3. *Digg now uses a “digg it” button to immediately record a “digg” within the same page*

Kevin Rose has noted that changing to a one-step process had a huge impact on the number of stories being dug and thus on the success of his site. Here is his view on what transpired:

> *There was a huge shift in activity on Digg when we made the move to the one-click digg in November 2005. Once we added Ajax, activity went through the roof on [the number of] diggs. It was just insane. Just the ease of the “one-click and you’re done” made all the difference in the world. Once the users grasped that the content is syndicated to friends, friends’ activities then went through the roof. These small incremental steps in feature additions drove the growth.**

Did you get that? Changing to a one-click digg was not technically difficult. Yet it made a significant impact on Digg’s success. It not only increased the ease of digging stories, but it also had an impact on other site activities. This is the power of making interactions lightweight.

Next, in Chapter 4, we look at a set of patterns that can make our interactions lightweight:

Contextual Tools
    Tools placed in context within the page content.

* From Rose’s talk at the San Francisco Bay Area Chapter of ACM SIGCHI (BayCHI), April 11, 2006. See *http://www.baychi.org/calendar/20060411/*.

Chapter 4
Contextual Tools

Interaction in Context

Most desktop applications separate functionality from data. Menu bars, toolbars, and palettes form islands of application functionality. Either the user chooses a tool to use on the data or makes a selection and then applies the tool.

Early websites were just the opposite. They were completely content-oriented. Rich tool sets were not needed for simply viewing and linking to content pages. Even in e-commerce sites like Amazon or eBay, the most functionality needed was the hyperlink and "Submit" button.

However, this simplistic approach no longer exists in the current web application landscape. As the Web has matured, a wide variety of application styles has emerged.

On one end of the spectrum there are simple sites that need no more functionality than the hyperlink and a "Submit" button. On the other end of the spectrum there are full applications hosted as a website. Google Search and Yahoo! Mail are two typical applications that illustrate this variation (Figure 4-1).

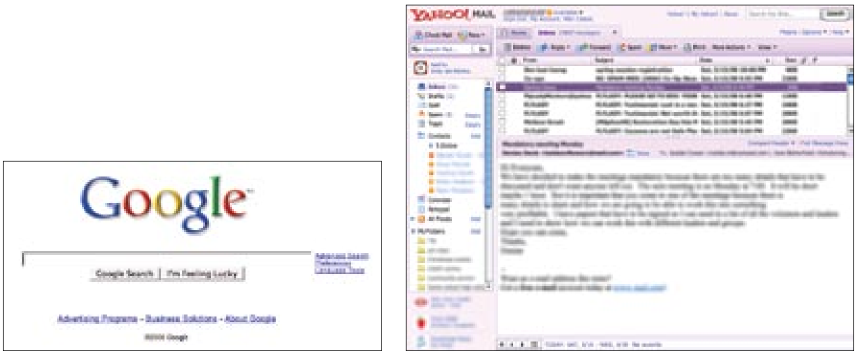


Figure 4-1. *Google Search needs only hyperlinks and a search button; Yahoo! Mail, on the other hand, is a full-featured application with toolbars and menus*

Between these two opposites are a lot of sites that need to mix content and functionality. It is to this intersection that we turn our attention in this chapter.

Think for a moment where user interfaces are headed.

Touch-based interfaces were the stuff of research labs and, more recently, interesting YouTube videos. But now they're as close as our phones. Most notably, the Apple iPhone brought touch to the masses (Figure 4-2). Gesture-based interfaces seemed even further out. Yet these became reality with the Nintendo Wii.

Figure 4-2. *The Apple iPhone introduced touch-based interfaces to the consumer market*

With both gesture and touch-based interfaces, interaction happens directly with the content.

Tip

The *content is the interface*. Instead of being contained in separate areas of functionality, the actions feel close to the objects being interacted with.

This concept also informs our current challenge. How do we bring tools nearer to the content to make the interaction as lightweight as possible?

Fitts's Law

Fitts's Law is an ergonomic principle that ties the size of a target and its contextual proximity to ease of use. Bruce Tognazzini restates it simply as:

The time to acquire a target is a function of the distance to and size of the target.

In other words, if a tool is close at hand and large enough to target, then we can improve the user's interaction. Putting tools in context makes for lightweight interaction.

Contextual Tools

We could simply isolate our functionality into islands of tools (toolbars and menus). But this would work against Fitts's Law by requiring more effort from the user. It would also add more visual weight to the page. Instead of interacting with the functionality separately, we can bring the functionality into the content with **Contextual Tools**.

Contextual Tools are the Web's version of the desktop's right-click menus. Instead of having to right-click to reveal a menu, we can reveal tools in context with the content. We can do this in a number of ways:

Always-Visible Tools
Place **Contextual Tools** directly in the content.

Hover-Reveal Tools
Show **Contextual Tools** on mouse hover.

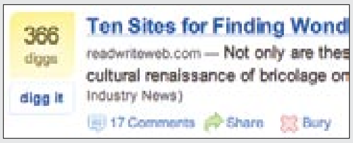
Toggle-Reveal Tools
A master switch to toggle on/off **Contextual Tools** for the page.

Multi-Level Tools
Progressively reveal actions based on user interaction.

Secondary Menus
Show a secondary menu (usually by right-clicking on an object).

Always-Visible Tools

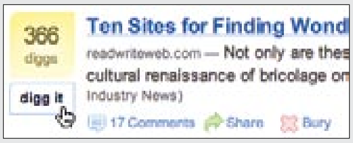
The simplest version of **Contextual Tools** is to use **Always-Visible Tools**. Digg is an example of making Contextual Tools always visible (Figure 4-3).


366
diggs
digg it
Ten Sites for Finding Wond
readwriteweb.com — Not only are thes
cultural renaissance of bricolage on
Industry News)
17 Comments
Share
Bury

Visible tool

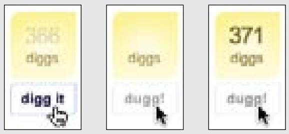
Beside each story is a digg scorecard. Just below is the "digg it" button. The digg button shows for all stories.

Other actions are represented less prominently.


366
diggs
digg it
Ten Sites for Finding Wond
readwriteweb.com — Not only are thes
cultural renaissance of bricolage on
Industry News)
17 Comments
Share
Bury

Invitation

On mouse hover, the digg button border changes to a darker color and the text label changes to black. Highlighting is an effective way to signal interactivity.



Completion

Once the user clicks the "digg it" button, the vote is counted. The current vote fades out and then the new digg count (including your vote) appears instantly. The digg button changes to "dugg" and is no longer clickable (indicated by the gray text).

Figure 4-3. *Digg's "digg it" button is a simple Contextual Tool that is always visible*

Considerations

The "digg it" button and Digg scorecard provide **Always-Visible Tools** next to each story.

Clear call to action

Why not hide the tools and only reveal them when the mouse is over the story? Since digging stories is central to the business of Digg, always showing the tool provides a *clear call to action*. There are other actions associated with news stories (comments, share, bury, etc.) but they are represented less prominently. In the case of Digg, the designers chose to show these at all times. An alternate approach would be to hide them and show them on mouse hover (we will discuss this approach in the next section).

It turns out that voting and rating systems are the most common places to make tools always visible. Netflix was the earliest to use a one-click rating system (Figure 4-4).

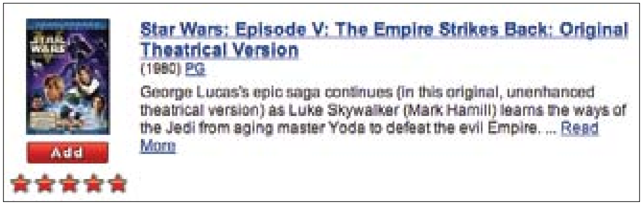

Star Wars: Episode V: The Empire Strikes Back: Original Theatrical Version
(1980) PG
George Lucas's epic saga continues (in this original, unenhanced theatrical version) as Luke Skywalker (Mark Hamill) learns the ways of the Jedi from aging master Yoda to defeat the evil Empire. ... Read More

Figure 4-4. *Netflix star ratings are always visible*

Just as with Digg, rating movies is central to the health of Netflix. The Cinematch™ recommendation engine is driven largely by the user's ratings. So a clear call to action (to rate) is important. Not only do the stars serve as a strong call to action to rate movies, but they also provide important information for the other in-context tool: the "Add" button. Adding movies to your movie-shipping queue is key to having a good experience with the Netflix service.

Relative importance

One way to clarify this process is to decide on the relative importance of each exposed action. Is the “digg it” action as important as the “bury it” action? In the case of Digg, the answer is no. The “digg it” action is represented as a button and placed prominently in the context of the story. The “bury it” action is represented as a hyperlink along with other “minor” actions just below the story. The contrast of a button and a hyperlink as well as its placement gives a strong indication as to the relative importance of each action.

Tip

If an action is critical, expose it directly in the interface.

Discoverability

Discoverability is a primary reason to choose **Always-Visible Tools**. On the flip side, it can lead to more visual clutter. In the case of Digg and Netflix, there is a good deal of visual space given to each item (story, movie). But what happens when the items you want to act on are in a list or table?

Generally **Contextual Tools** in a list work well when the number of actions is kept to a minimum. Gmail provides a single **Always-Visible Tool** in its list of messages—the star rating—for flagging emails (Figure 4-5).

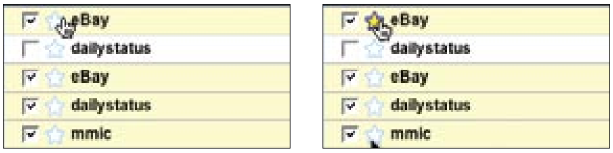
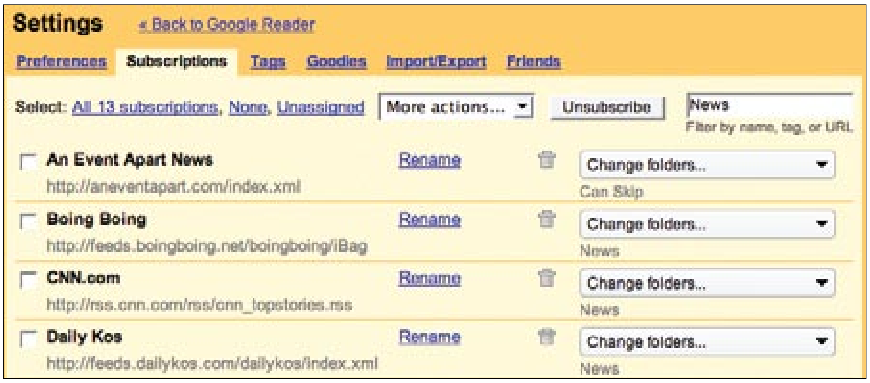


Figure 4-5. *Google Mail uses Contextual tools to flag favorites*

Simply clicking the star flags the message as important. The unstarred state is rendered in a visually light manner, which minimizes the visual noise in the list.

The following list, from Google Reader, takes a different approach. It shows several tools for managing subscriptions: rename, delete, and change folders for each subscription in the list. This is convenient but is definitely heavier visually (Figure 4-6).


Settings « Back to Google Reader

Preferences Subscriptions Tags Goodies Import/Export Friends

Select: All 13 subscriptions, None, Unassigned | More actions... | Unsubscribe | News — Filter by name, tag, or URL

An Event Apart News — http://aneventapart.com/index.xml — Rename — Change folders... — Can Skip

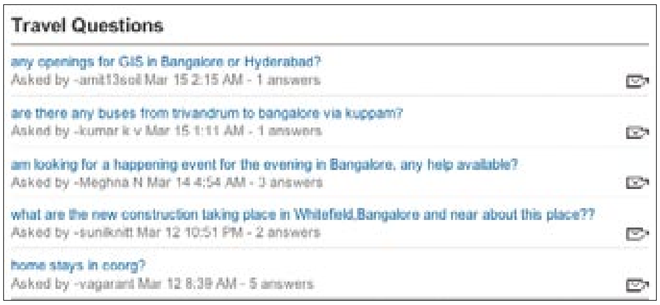
Boing Boing — http://feeds.boingboing.net/boingboing/iBag — Rename — Change folders... — News

CNN.com — http://rss.cnn.com/rss/cnn_topstories.rss — Rename — Change folders... — News

Daily Kos — http://feeds.dailykos.com/dailykos/index.xml — Rename — Change folders... — News

Figure 4-6. *Google Reader's Manage Subscriptions page displays lots of actions for each subscription, leading to a visually heavier display*

Sometimes concerns over visual noise must take a back seat to discoverability. The Yahoo! India Our City team struggled with a design early on. They wanted to hide the "email this" icon and only show it on hover. However, since the site was specifically for India, they were concerned with how much exposure the population had with simple web interactions like mouse rollover. So instead of hiding the icon, they chose to show it for every story (Figure 4-7).


Travel Questions

any openings for GIS in Bangalore or Hyderabad?
Asked by -amit13soil Mar 15 2:15 AM - 1 answers

are there any buses from trivandrum to bangalore via kuppam?
Asked by -kumar k v Mar 15 1:11 AM - 1 answers

am looking for a happening event for the evening in Bangalore, any help available?
Asked by -Meghna N Mar 14 4:54 AM - 3 answers

what are the new construction taking place in Whitefield,Bangalore and near about this place??
Asked by -sunilknitt Mar 12 10:51 PM - 2 answers

home stays in coorg?
Asked by -vagarant Mar 12 8:38 AM - 5 answers

Figure 4-7. *Yahoo! India Our City was designed for users who were not familiar with mouse rollover; the email icon is shown at all times*

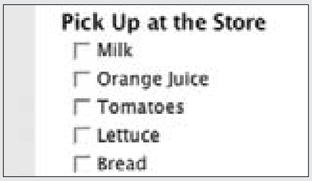
Best Practices for Always-Visible Tools

Here are some best practices to keep in mind:

- Make your **Contextual Tools** always visible if it is important to make a prominent call to action.
- Keep visual clutter to a minimum.
- Keep the number of visual items to a minimum.

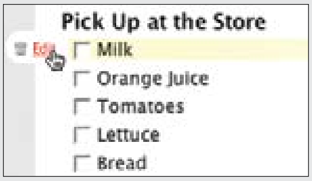
Hover-Reveal Tools

Instead of making **Contextual Tools** always visible, we can show them on demand. One way to do this is to reveal the tools when the user pauses the mouse over an object. The **Hover-Reveal Tools** pattern is most clearly illustrated by 37 Signal's Backpackit (Figure 4-8). To-do items may be deleted or edited directly in the interface. The tools to accomplish this are revealed on mouse hover.



Normal state

The edit and delete tools are hidden in the normal state.



Invitation

On mouse hover, the tools are revealed. The tools are "cut" into the gray bar, drawing the eye to the change.

Figure 4-8. *Backpackit reveals its additional tools on mouse hover*

Considerations

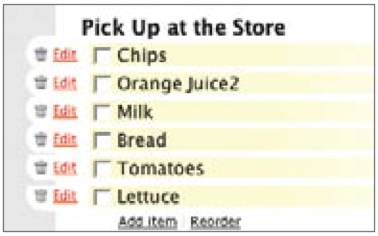
The gray bar on the left is a nice visual reinforcement for the interaction. By allowing the tools to "cut" into the sidebar, the designers draw your eye to the available tools. The light yellow background draws attention to the to-do item being acted on. These two simple treatments make it clear which line has the focus and that additional tools have been revealed.

Tip

To reduce visual clutter, hide non-primary actions until they are needed.

Visual noise

Showing the items on hover decreases the visual noise in the interface. Imagine if instead the delete and edit actions were always shown for all to-do items. Figure 4-9 shows just how visually noisy that approach would have been.



Pick Up at the Store

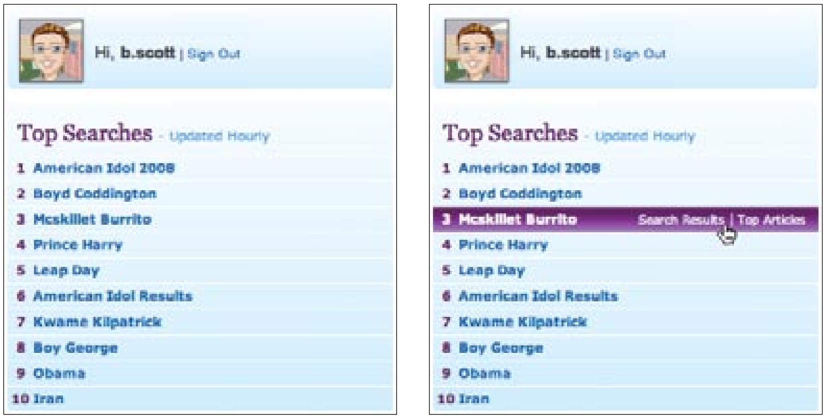
- Edit Chips
- Edit Orange Juice2
- Edit Milk
- Edit Bread
- Edit Tomatoes
- Edit Lettuce

Add item | Reorder

Figure 4-9. *What the Backpackit interface would have looked like if the Contextual Tools were always visible*

Yahoo! Buzz reveals its tools on mouse hover for both its Top Searches (Figure 4-10) and Top Stories (Figure 4-11).

For Top Searches, it is important to keep the top-ten list as simple as possible. Showing tools would compete with the list itself. Since the actions "Search Results" and "Top Articles" (Figure 4-10, right) are less important, they are revealed on hover. The actions may be important, but making the content clear and readable is a higher priority.



Hi, b.scott | Sign Out

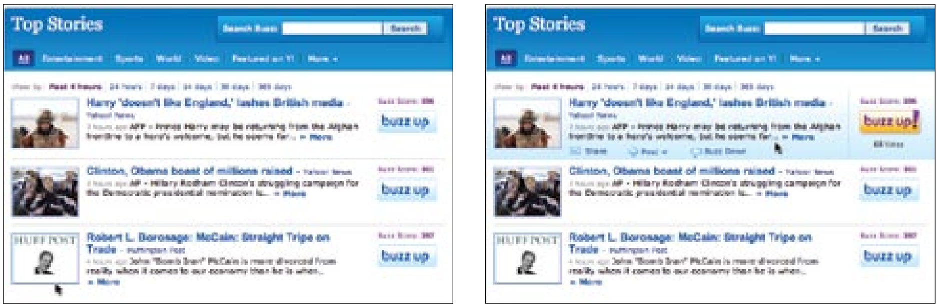
Top Searches - Updated Hourly

1 American Idol 2008
2 Boyd Coddington
3 Mcskillet Burrito
4 Prince Harry
5 Leap Day
6 American Idol Results
7 Kwame Kilpatrick
8 Boy George
9 Obama
10 Iran

3 Mcskillet Burrito Search Results | Top Articles

Figure 4-10. *Yahoo! Buzz reveals additional tools for the top searches when the user hovers over each item*

Similarly, for Top Stories, Yahoo! Buzz shows only "Share", "Post", and "Buzz Down" tools on hover. "Buzz Up" is shown at all times, but gets extra visual treatment on mouse hover (Figure 4-11, right). "Buzz Up" is important enough to show at all times, but can be toned down when not the focus.



Harry 'doesn't like England,' lashes British media

Clinton, Obama boast of millions raised

Robert L. Borosage: McCain: Straight Tripe on Trade

Figure 4-11. *Yahoo! Buzz highlights the row and brings in additional tools*

Discoverability

A serious design consideration for **Hover-Reveal Tools** is just how discoverable the additional functionality will be. In the earlier Backpackit example (Figure 4-8), while the **Contextual Tools** are revealed on hover, the checkbox is always visible for each to-do item. To check off an item, users have to move the mouse over it. When they do, they will discover the additional functionality.

Flickr provides a set of tools for contacts. To avoid clutter, contact profile photos are shown without any tool adornment. When the mouse hovers over the contact's photo, a drop-down arrow is revealed (Figure 4-12). Clicking reveals a menu with a set of actions for the contact. This works because users often know to click on an image to get more information. Being drawn to the content is a good way to get the user to move the mouse over the area and discover the additional functionality.

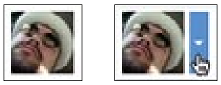


Figure 4-12. *Flickr reveals the drop-down menu on hover*

Tip

Help users understand revealed tools by using familiar idioms (such as hyperlinks for actions or drop-down arrows to expose additional functionality).

Yahoo! Mail's flagging feature is revealed when the user hovers over the flagged column on a mail message (Figure 4-13). Contrast this to Google's always-revealed star approach we discussed earlier (Figure 4-5).

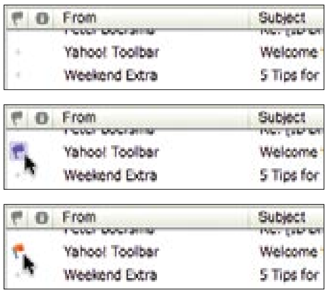


Figure 4-13. *Yahoo! Mail reveals the flag button when the user's mouse hovers over the flag column of a message*

The Yahoo! approach is visually cleaner, but less discoverable. We will have more to say about making tools discoverable in Chapter 10 when we discuss **Dynamic Invitations**.

Contextual Tools in an overlay

Sometimes there are several actions available for a focused object. Instead of placing tools beside the object being acted on, the revealed tools can be placed in an overlay. However, there can be issues with showing contextual tools in an overlay:

1. Providing an overlay feels heavier. An overlay creates a slight contextual switch for the user's attention.
2. The overlay will usually cover other information—information that often provides context for the tools being offered.
3. Most implementations shift the content slightly between the normal view and the overlay view, causing the users to take a moment to adjust to the change.
4. The overlay may get in the way of navigation. Because an overlay hides at least part of the next item, it becomes harder to move the mouse through the content without stepping into a "landmine."

Anti-pattern: Hover and Cover

Figure 4-14 illustrates all four of these situations. In an early version of Yahoo! for Teachers,[*] hovering over a clipped item brought in three tools: copy, delete, and preview. However, when these tools were placed in an overlay, it covered the item to the right, making it hard to see that content and even navigate to it. In addition, since the overlay had some additional padding (as well as rounded corners), the image shown in the overlay was about two pixels off from the non-overlay version. This slight jiggle was distracting. To add insult to injury, the overlay was sluggish to bring into view.

[*] Yahoo! for Teachers was only released in beta and never widely publicized. It was recently closed down, and another company (edtuit.com) will be launching a similar site.

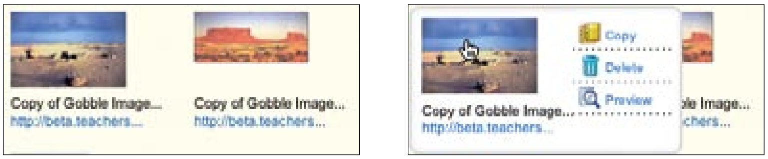


Figure 4-14. *An early version of the Yahoo! for Teachers beta revealed Contextual Tools in an overlay; the overlay covered more than half of the item to its right*

The final straw was if users wanted to delete several items, they would hover over the image, wait for the overlay, click Delete, then be forced to move out and back in again to activate the next image's **Contextual Tools** (Figure 4-15). **Hover and Cover** is a common anti-pattern that occurs when exposing an overlay on hover and hiding important context or further navigation.

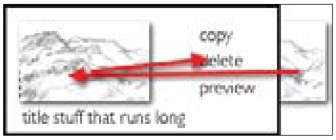


Figure 4-15. *Navigating required a zigzag approach to get around the tool overlay*

Hover and Cover was resolved by no longer using an overlay. Instead, additional margin space was added to each image, and the **Contextual Tools** were hidden. On mouse hover, the tools were simply revealed, along with a border defining the image being acted on (Figure 4-16).

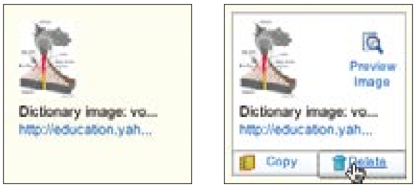


Figure 4-16. *In the redesigned version, tools were shown on hover directly surrounding the image instead of in an overlay*

The difference (Figure 4-14 versus Figure 4-16) is dramatic. Not only is the experience improved, but overall page performance is improved as well. On mouse hover, the image no longer shifts in a distracting manner. In redesign, the delete always shows up in the same place relative to the image; this means the user "remembers spatially" where the command is, making it easier to target (Figure 4-17).

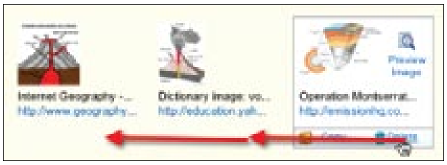


Figure 4-17. *Since the Delete action appears in the same spot relative to the image, the user can easily anticipate where the action will appear*

Tip

Be careful when using overlays to expose additional information or tools. The overlay can get in the way of normal navigation or hide important information.

Anti-pattern: Mystery Meat

Have you ever found a can in the back of the pantry whose label has long since fallen off? The only way to identify this *mystery meat* is to open it. Unidentifiable icons are pretty much the same as a row of unlabeled cans. You have to hover over each icon and hope for a tool tip to label it. Even worse is when no tool tip is available. The easiest way to avoid this predicament is to use either a text label or combine an icon with a text label. **Mystery Meat** is a common anti-pattern that occurs when you have to hover over an item to understand how to use it.

Tip

Don't make users hover over your tools in order to figure out what they mean.

Figure 4-18 illustrates this in an early version of Zooomr. The only recourse for the user was to pause over each icon and wait a second or so to read a tool tip about the purpose of the icon. This does not create a lightweight interaction!

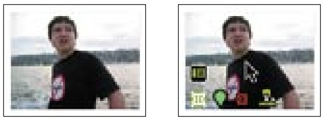


Figure 4-18. *A very early version of Zooomr revealed a number of unidentifiable icons when the mouse hovered over a photo*

Activation

Tool overlays should activate immediately. The tools are an extension of the interface, and any delay creates too much of a delay between the activation and invocation of the action. In Chapter 5, we will discuss **Dialog Overlays**. In that discussion we suggest a delay before showing informational overlays. Why the difference? Since information may not be needed to understand the object, and given the fact that activation might be accidental, it is best to place a small delay when showing the additional information (usually a half-second delay is sufficient). But actions are different. Following the suggestions just mentioned (avoid **Hover and Cover** anti-pattern), the actions can be revealed without a lot of disruption. And if they show immediately, the user can access the additional commands almost as quickly as with **Always-Visible Tools**.

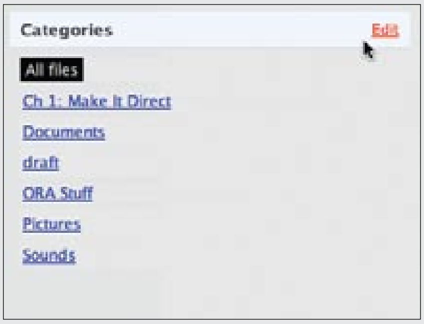
Best Practices for Hover-Reveal Tools

Here are some best practices to keep in mind:

- Hide **Contextual Tools** behind a mouse hover if the actions are secondary and you want to make the content more important (or other visible commands more important).
- Avoid using overlays when revealing additional tools. They will lead to the **Hover and Cover** anti-pattern, as well as require the user to perform mouse gymnastics to accomplish the most basic tasks.
- When additional tools are revealed, make sure that all parts of the page remain stable.
- Avoid any element shifting by a few pixels or page elements moving around. These cause the user to be directed away from what is really happening.
- Make sure revealed icons are clear and understandable. When possible, just use text labels.
- Activate tool overlays instantly. Unlike informational overlays, the user needs the additional tools to be available for immediate interaction.

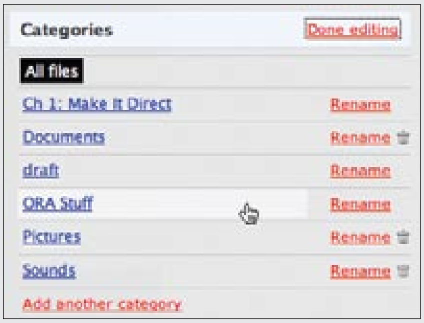
Toggle-Reveal Tools

A variation on the two previous approaches is to not show any **Contextual Tools** until a special mode is set on the page. A good example of **Toggle-Reveal Tools** is in Basecamp's category editing, which we discussed in Chapter 1 (Figure 4-19).



Not visible

Each category is listed in this section. The "Edit" link at the top is the way to edit the category section.



Visible in edit mode

Each category gets a "Rename" link and where appropriate a trashcan is displayed (for empty categories that may be deleted).

This is a "soft" mode, since the user can ignore the additional tools and choose to do something different on the page.

Figure 4-19. *Basecamp reveals category-editing tools only when the edit mode is turned on for the area*

Considerations

Here are a few considerations to keep in mind when using **Toggle-Reveal Tools**.

Soft mode

Generally, it is a good thing to avoid specific modes in an interface. However, if a mode is *soft* it is usually acceptable. By "soft" we mean the user is not trapped in the mode. With Basecamp, the user can choose to ignore the tools turned on. It just adds visual noise and does not restrict the user from doing other actions. This is a nice way to keep the interaction lightweight.

Tip

Interfaces should strive to be modeless. Often, though, a "soft mode" can be employed to provide context for an action that is easy to activate and easy to remove.

When would you use this technique? When the actions are not the main thing and you want to reduce visual noise. This fits the category example perfectly. Items are renamed or deleted occasionally. It is common, however, to want to click through and see the contents of a category (the category is always hyperlinked). Hence, make it readable and easily navigable in the normal case—but still give the user a way to manage the items in context.

Google Reader could potentially be improved in this manner. In the current interface, clicking "Manage Subscriptions" takes the user to another page to edit subscriptions. One possible change is the addition of an "edit" button that toggles in a set of context tools for each subscription (Figure 4-20). This would allow the user to rename and unsubscribe without leaving the context of the reading pane.

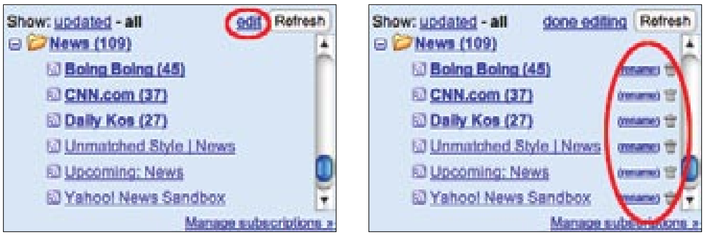


Figure 4-20. *Adding an "edit" link to Google Reader's feed list and toggling in common actions could potentially make it easier to manage subscriptions*

Best Practices for Toggle-Reveal Tools

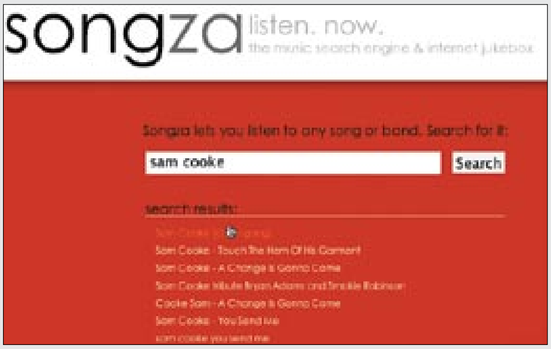
Here are some best practices to keep in mind:

- Toggle a tool mode for an area or page when the actions are not the main flow, but you want to provide the most direct way to act on these objects when the need arises.
- Make the activation and deactivation of the edit mode as symmetrical as possible.
- Keep the transition between display and edit as seamless as possible to provide a "soft mode" for editing.

Multi-Level Tools

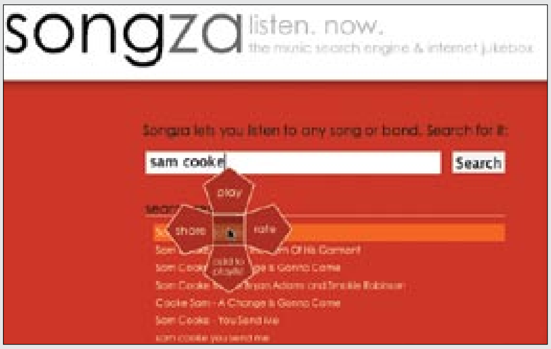
Contextual Tools can be revealed progressively with **Multi-Level Tools**. Songza[*] provides a set of tools that get revealed after a user clicks on a song. Additional tools are revealed when hovering over the newly visible tools (Figure 4-21).

[*] Aza Raskin is the designer of Songza, founder of Humanized. He is the son of the late human-computer interface expert Jef Raskin.


songza
listen. now.
the music search engine & internet jukebox
Songza lets you listen to any song or band. Search for it:
sam cooke
Search
search results:
Sam Cooke - Touch The Hem Of His Garment
Sam Cooke - A Change Is Gonna Come
Sam Cooke tribute Bryan Adams and Smokie Robinson
Cooke Sam - A Change Is Gonna Come
Sam Cooke - You Send Me
sam cooke you send me

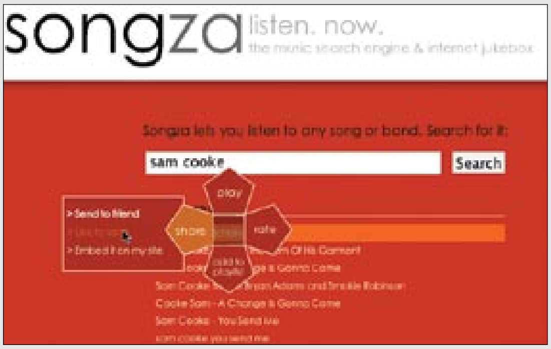
Normal state

The tools are not visible normally. Mouse hover just highlights the song—it does not reveal the **Contextual Tools**.



Click activation

On mouse click, a cloverleaf-style menu is shown with the four basic functions: play, rate, add, and share.



Hover expose

Second-level actions are exposed while hovering over share or rate.

Figure 4-21. *Songza uses a multi-level contextual tool menu*

Considerations

Songza reveals the four options "play", "rate", "share", and "add to playlist" after the user clicks on a song title. Hovering over "share" or "rate" reveals a secondary set of menu items (Figure 14-21, center).

Radial menus

Radial menus[*] such as in Songza have been shown to have some advantages over more traditional menus. First, experienced users can rely on muscle memory rather than having to look directly at the menu items. Second, the proximity and targeting size make the menu easy to navigate since the revealed menu items are all equally close at hand (recall Fitts's Law).

The one potential downside to this approach is that rating a song requires several steps: an initial click on the song, moving the mouse over the "rate" menu item, then clicking either the thumbs up or thumbs down option. If rating songs was an important activity, the extra effort might prevent some users from doing so. An alternate approach would be to replace "rate" directly with the thumbs up and the thumbs down options.

Activation

Another interesting decision Songza made was to not activate the radial menu on hover. Instead, the user must click on a song to reveal the menu. Activating on click makes the user intent more explicit.

Making activation more explicit avoids the issues described earlier in the **Hover and Cover** anti-pattern. The user has chosen to interact with the song. Conversely, with a mouse hover, it's never quite clear if the user meant to activate the menu or just happened to pause over a song title.

Default action

However, this does mean there is no way to start a song playing with just a simple click. Playing a song requires moving to the top leaf. One possible solution would be to place the "play" option in the middle of the menu (at the stem) instead of in one of the leaves. Clicking once would activate the menu. Clicking a second time (without moving the mouse) would start playing the song. This interaction is very similar to one commonly used in desktop applications: allowing a double-click to activate the first item (default action) in a right-click menu.

Tip

Keep the most common actions as near to the activation point as possible.

[*] Also known as *pie menus*. See Jack Callahan et al. (1988), "An empirical comparison of pie vs. linear menus." *Proceedings of ACM CHI Conference on Human Factors in Computing Systems*: 95–100.

Contextual toolbar

Picnik is an online photo-editing tool that integrates with services like Flickr. In all, there are six sets of tools, each with a wide range of palette choices. Picnik uses **Multiple-Level Tools** to expose additional functionality. By wrapping the photo with tools in context and progressively revealing the levels of each tool, Picnik makes editing straightforward (Figure 4-22).

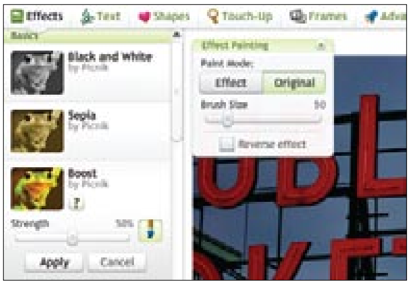
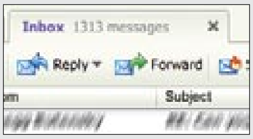


Figure 4-22. *Picnik wraps layers of Contextual Tools around the image being edited*

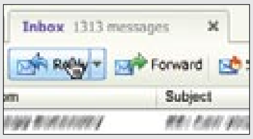
Muttons

Another variation on **Multi-Level Tools** is the "mutton" (menu + button = mutton). Muttons are useful when there are multiple actions and we want one of the actions to be the default. Yahoo! Mail uses a mutton for its "Reply" button (Figure 4-23).



Normal state

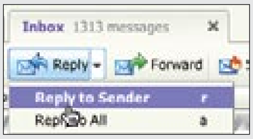
Yahoo! Mail displays the "Reply" mutton in its toolbar as a button with a drop-down arrow control.



As a button

On mouse hover, the button gets a 3D treatment and color highlight. The drop-down arrow gets the same treatment to call out its functionality.

Clicking the "Reply" button at this point will trigger a reply without activating the menu.



As a menu

Clicking on the drop-down arrow reveals two commands: "Reply to Sender" is the same as the default "Reply" button action; "Reply to All" is an additional action that was hidden until the menu was revealed.

Figure 4-23. *Yahoo! Mail's "Reply" button looks like a drop-down when hovered over; clicking "Reply" replies to sender, and clicking the drop-down offers the default action as well as "Reply to All"*

Clicking “Reply” performs the individual reply. To reply to all, the menu has to be activated by clicking on the drop-down arrow to show the menu.

Muttons are used to:

- Provide a default button action (“Reply to Sender”)
- Provide a clue that there are additional actions
- Provide additional actions in the drop-down

If muttons are not implemented correctly, they can be problematic for those using accessibility technologies. Because an earlier version of Yahoo! Mail did not make the mutton keyboard accessible, Yahoo!’s accessibility guru, Victor Tsaran, was convinced that there was no “Reply to All” command in the Yahoo! Mail interface. Only after the mutton was made more accessible could he find the “Reply” command.

Anti-pattern: Tiny Targets

At the beginning of this chapter, we discussed Fitts’s Law. Recall that the time it takes to acquire a target is a function of both distance and size. Even if tools are placed close by in context, don’t forget to make them large enough to target.

Both Flickr and Yahoo! Mail provide a reasonable-size target for the drop-down arrow. Compare this with the expand/collapse arrow used in an early version of Yahoo! for Teachers (Figure 4-24).

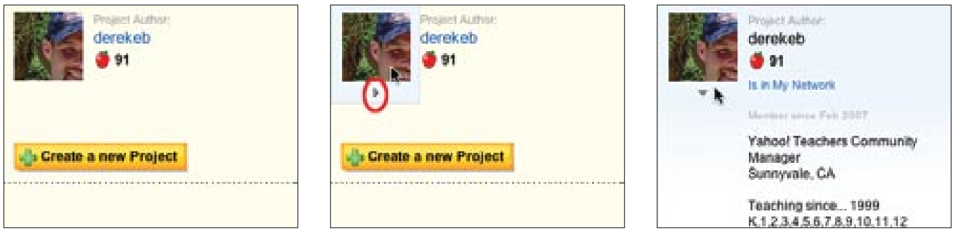


Figure 4-24. *The Yahoo! for Teachers Profile Card info is hidden unnecessarily*

Tip

Never use small targets. Make targets large enough to notice and interact with.

The arrow is tiny (8×8 pixels). It is exposed only on hover. Providing **Tiny Targets** makes interaction much more difficult. An alternate approach would be to always show a “more info” link. Clicking it could toggle the additional profile information. Alternatively, providing a larger target for the arrow would improve its findability and targeting.

Best Practices for Multi-Level Tools

Here are some best practices to keep in mind:

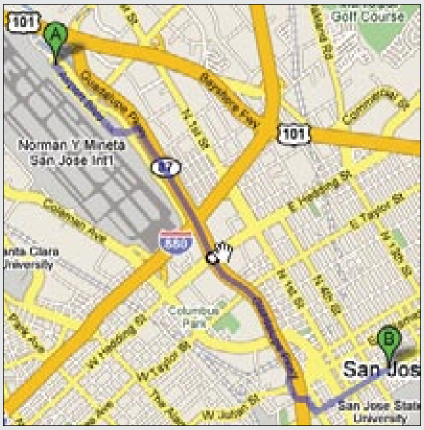
- Use **Multi-Level Tools** when you want to avoid revealing **Contextual Tools** on a mouse hover.
- Use **Multi-Level Tools** to make activation explicit.
- Use muttons when you have a default action that the user normally takes but alternate actions are still fairly frequent.
- Avoid cascades where possible. Users have a hard time maneuvering the various mouse turns that are required to get to these secondary menus.
- Keep actions as close to the activation point as possible.

Secondary Menu

Desktop applications have provided **Contextual Tools** for a long time in the form of **Secondary Menus**. These menus have been rare on the Web. Google Maps uses a secondary menu that is activated by a right-click on a route. It shows additional route commands (Figure 4-25).

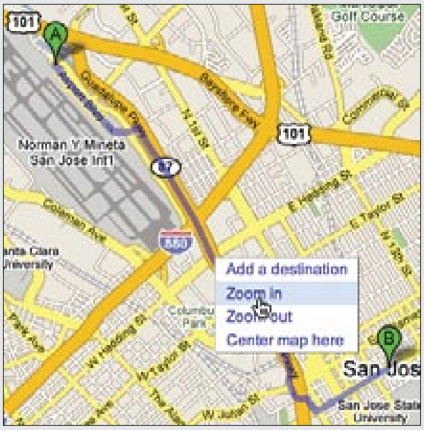
Normal view of route

Routes give no indication of additional functionality when not hovered over.



Invitation

When the mouse is over the route, potential stops are marked with a white circle.



Menu

Right-clicking on the item exposes four commands that act on the point selected: "Add a destination", "Zoom in", "Zoom out", and "Center map here".

Figure 4-25. *Google Maps uses a right-click menu to add new route stops or to adjust the map around the current point on the route*

Considerations

Secondary Menus have not been common in web applications.

Conflict with browser menu

One problem is the browser inserts its own right-click menu. Replacing the menu in normal content areas can confuse users, as they may not know if the standard browser menu or the application-specific menu will be shown. It will depend on whether it is clear that an object exists in the interface (as in the route line above), and if the menu is styled differently enough to disambiguate the menus.

Discoverability

As a general rule, never put anything in the **Secondary Menu** that can't be accomplished elsewhere. **Secondary Menus** are generally less discoverable. More advanced items or shortcuts, however, can be placed in the **Secondary Menu** as an alternate way to accomplish the same task.

Accessibility

Right-click is not the only way to activate a **Secondary Menu**. You can activate the menu by holding down the mouse for about one second. This provides a more accessible approach to popping up a **Secondary Menu**. This technique is used in the Macintosh Dock. Clicking and holding down on an application in the dock will reveal the **Secondary Menu** without requiring a right-click activation.

Acting on multiple objects

Keep in mind that all of the other **Contextual Tools** presented in this chapter have a limitation on the number of items they can operate on. **Always-Visible Tools**, **Hover-Reveal Tools**, **Toggle-Reveal Tools**, and **Multi-Level Tools** all operate on a single item at a time (even **Toggle-Reveal Tools** just shows a tool per item). **Secondary Menus** are different. They can be combined with a selection model (as described in Chapter 3) to perform actions on selected set of items.

Best Practices for Secondary Menus

Here are some best practices to keep in mind:

- Place alternate or shortcut commands in **Secondary Menus**.
- Consider activating **Secondary Menus** by holding down the mouse for one second as an alternative to right-clicking to show the menu.
- Style **Secondary Menus** differently than the Browser's standard secondary menu.
- Avoid **Secondary Menus** for all but redundant commands
- Avoid **Secondary Menus** in places where there is little resemblance to a traditional web page.
- Use **Secondary Menus** for operating on a selected set of objects.

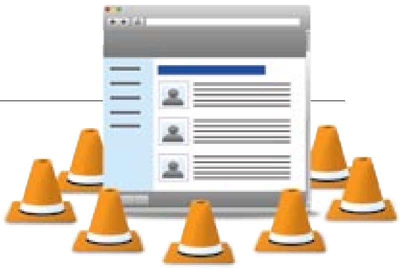
General Practices for Contextual Tools

To sum up, here are some overall best practices to keep in mind:

- **Contextual Tools** are useful for reducing the user's path to completing a task. By placing tools near the point of focus and making these tools easy to activate, you can create a lightweight interaction.
- Use **Contextual Tools** when you have no way to select elements and operate on them as a whole.
- Use **Contextual Tools** when you want to shorten the path the user must take to complete a task for an item shown on the page.
- Use **Contextual Tools** when you want to provide a clear call to action at the point of focus.
- Always make your actions as immediate as possible, avoiding additional steps where you can.
- Where possible use familiar techniques (hyperlinks, drop-down arrows, buttons) when providing actions in unexpected places (hover-revealed **Contextual Tools**).
- Make the actions clear and direct. Avoid using icons for all but the most obvious (an [x] or a trashcan for delete, for instance).
- Use buttons for strong calls to action and links for minor actions.
- Make sure that targets used to open menus or expand information are suitably sized. Never use 8×8 pixel targets.
- Make tools easy to understand, easy to target, and quick to execute. This will make them feel lightweight.

Principle Three

Stay on the Page



Flow

In the book, *Flow: The Psychology of Optimal Experience,* published by Harper Perennial, Mihaly Csikszentmihalyi describes the state of "optimal experience" as *the times when people enter a state of concentration so focused it creates a state of effortless control*. Athletes may enter into flow and report the feeling of unself-consciousness as well as rising to the peak of their abilities. Flow, however, can be broken by a sudden awareness of the surroundings or by some interruption that happens to come along.

Unfortunately, users of our web applications rarely experience this level of happiness. In fact, the traditional web experience is punctuated with a page refresh each time the user chooses an action. It's like watching a play where the curtain comes down between each line of dialogue. The page refresh creates an artificial break in the action—or a break in the user's flow.

Change Blindness

The break can cause visual consequences as well. I recently took some of my children to the Exploratorium in San Francisco (a wonderful hands-on science museum for all ages). An exhibit that caught my eye was the one demonstrating *change blindness*.* A large screen displayed an image of a store-front typical of those seen in most urban areas, complete with awning, windows, doors—all of a distinctive style. Then suddenly a new updated image of the store-front replaced the original one. The new image had a slight change from the original. However, try as I might I could not detect the change. Why? The transition was punctuated by a very brief (less than a fourth of a second) delay. Showing the original image, going blank, then showing the second image made it really hard to detect the change between the two.

Wikipedia describes *change blindness* as:

> *In visual perception, change blindness is the phenomenon where a person viewing a visual scene apparently fails to detect large changes in the scene.*†

* You can see a demonstration of this at *http://www.usd.edu/psyc301/ChangeBlindness.htm*.

† See *http://en.wikipedia.org/wiki/Change_blindness*.

Fortunately the exhibit included a button to press that removed the “page refresh.” When I held down this button, the change was obvious.

Including the page refresh in web applications was not the desire of web designers but rather an artifact of the technology underlying the Web. Each action generated a request for a new page, which included the refresh break. With this limitation, the ability to provide a seamless flow-based experience with continuous visual perception to the user was almost impossible.

However, this is no longer the case. We now have a button we can hold down! With the rise of Ajax, Flash, and other new technologies, it is possible to perform actions and bring back results for those actions while remaining on the same page and disturbing none of the surrounding context.

The principle *Stay on the Page* gets at the idea of creating a continuous visual perception that enhances flow nirvana. Given the history of the Web, it is important to always ask ourselves, “Can we create this experience in context, within the current page?” Sometimes, though, we may answer these questions with a “No.” That can be OK. It is important to realize that there are times when switching the page makes more sense than staying within the page. We will explore these situations in the next few chapters.

In Chapters 5 through 8, we look at four ways to keep the user on the page:

Overlays
: Instead of going to a new page, a mini-page can be displayed in a lightweight layer over the page.

Inlays
: Instead of going to a new page, information or actions can be inlaid within the page.

Virtual Pages
: By revealing dynamic content and using animation, we can extend the virtual space of the page.

Process Flow
: Instead of moving from page to page, sometimes we can create a flow within a page itself.

Chapter 5

Overlays

Overlays are really just lightweight pop ups. We use the term *lightweight* to make a clear distinction between it and the normal idea of a *browser pop up*. Browser pop ups are created as a new browser window (Figure 5-1). *Lightweight overlays* are shown within the browser page as an overlay (Figure 5-2). Older style browser pop ups are undesirable because:

- Browser pop ups display a new browser window. As a result these windows often take time and a sizeable chunk of system resources to create.
- Browser pop ups often display browser interface controls (e.g., a URL bar). Due to security concerns, in Internet Explorer 7 the URL bar is a permanent fixture on any browser pop-up window.

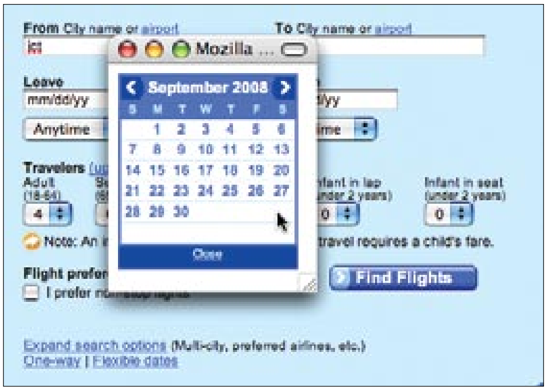

From City name or airport
To City name or airport
Mozilla ...
September 2008
S M T W T F S
Leave
mm/dd/yy
Anytime
Travelers
Adult (18-64)
Infant in lap (under 2 years)
Infant in seat (under 2 years)
Note: An ... travel requires a child's fare.
Close
Flight prefer...
I prefer n...
Find Flights
Expand search options (Multi-city, preferred airlines, etc.)
One-way | Flexible dates

Figure 5-1. *If Orbitz used a browser pop-up window for its calendar chooser (it does not), this is how it might look*

By using either Flash or Ajax-style techniques (Dynamic HTML), a web application can present a pop up in a lightweight overlay within the page itself. This has distinct advantages:

- Lightweight overlays are just a lightweight in-page object. They are inexpensive to create and fast to display.
- The interface for lightweight overlays is controlled by the web application and not the browser.
- There is complete control over the visual style for the overlay. This allows the overlay to be more visually integrated into the application's interface (compare Figures 5-1 and 5-2).

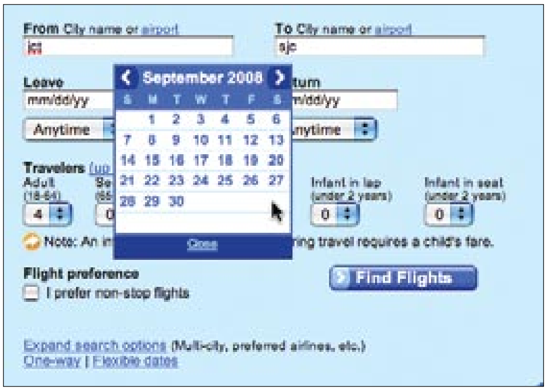


Figure 5-2. *Orbitz uses a lightweight DHTML overlay for its calendar chooser; since it does not require the overhead of a separate browser window, it can pop up quickly and is better integrated into the page visually*

Lightweight overlays can be used for asking questions, obtaining input, introducing features, indicating progress, giving instructions, or revealing information. They can be activated directly by user events (e.g., clicking on an action, hovering over objects) or be provided by the web application at various stages in the completion of an action.

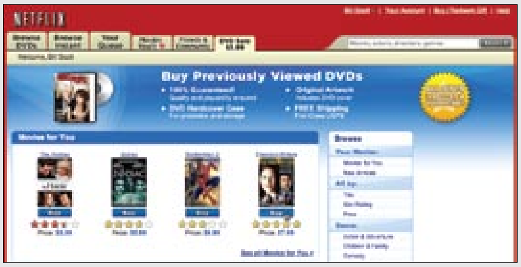
Tip

Never use browser overlays. They are expensive to create, hard to control, slower than lightweight overlays, and visually unappealing.

We will look at three specific types of overlays: **Dialog Overlays**, **Detail Overlays**, and **Input Overlays**.

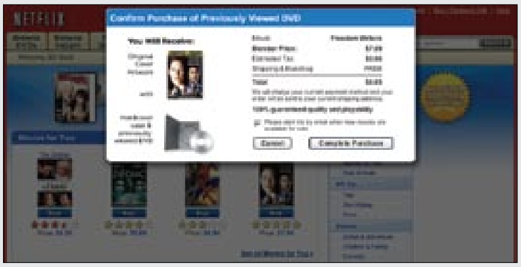
Dialog Overlay

Dialog Overlays replace the old style browser pop ups. Netflix provides a clear example of a very simple **Dialog Overlay**. In the "previously viewed movies for sale" section, a user can click on a "Buy" button to purchase a DVD. Since the customer purchasing the DVD is a member of Netflix, all the pertinent shipping and purchasing information is already on record. The complete checkout experience can be provided in a single overlay (Figure 5-3).



Activation

Clicking the "Buy" button initiates the purchase process.



Overlay treatment

The confirmation dialog is shown in a lightweight overlay. Since the overlay is modal (interaction is only accepted in the overlay) the rest of the page is dimmed down. The user may also cancel the purchase.

Figure 5-3. *Netflix uses a lightweight pop up to confirm a previously viewed DVD purchase; in addition, it uses the Lightbox Effect to indicate modality*

Considerations

Because the overlay is a lightweight pop up, the confirmation can be displayed more rapidly and the application has complete control over its look and placement.

Lightbox Effect

One technique employed here is the use of a **Lightbox Effect**. In photography a lightbox provides a backlit area to view slides. On the Web, this technique has come to mean bringing something into view by making it brighter than the background. In practice, this is done by dimming down the background.

You can see the **Lightbox Effect** pattern used by Flickr when rotating images (Figure 5-4).

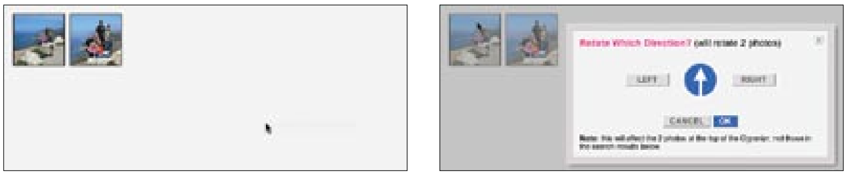


Figure 5-4. *Flickr also uses a Lightbox Effect to focus attention on the overlay*

The **Lightbox Effect** is useful when the **Dialog Overlay** contains important information that the user should not ignore. Both the Netflix Purchase dialog and the Flickr Rotate dialog are good candidates for the **Lightbox Effect**. If the overlay contains optional information, then the **Lightbox Effect** is overkill and should not be used.

Modality

Overlays can be modal[*] or non-modal. A modal overlay requires the user to interact with it before she can return to the application. In both the Netflix example (Figure 5-3) and the Flickr example (Figure 5-4), the overlays are *modal*: users cannot interact with the main Netflix or Flickr page until they perform the action or cancel the overlay. In both cases, modality is reinforced with the **Lightbox Effect**. Dimming down the background cues the user that this portion of the interface cannot be interacted with.

Sometimes overlays are non-modal. An example of this can be found in the Netflix site. When a DVD is added to the user's shipping list (queue), a confirmation overlay is shown (Figure 5-5). While it may appear that the only way to dismiss the overlay is by clicking the "Close" box in the upper-right corner, in reality the user can click anywhere outside the overlay (in the dimmed area) and the overlay will dismiss. In this case the **Lightbox Effect** is used to focus the user's attention on the confirmation and recommendations available.

* For a discussion on modality, see Jef Raskin's "The Humane Interface" (2000), 100, 141.

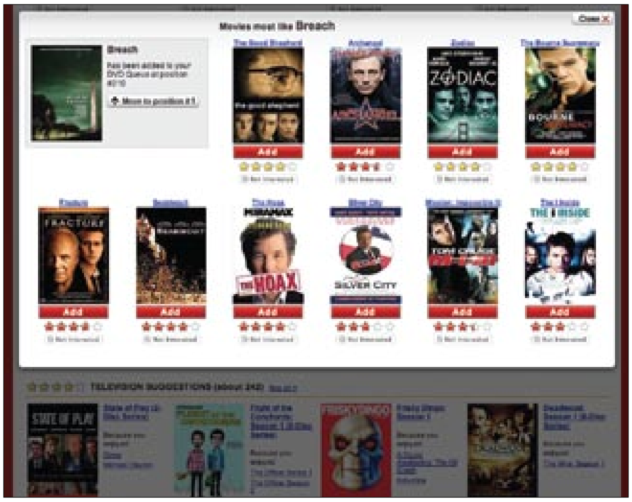


Figure 5-5. *Netflix uses a non-modal overlay with the Lightbox Effect to focus attention on the confirmation and recommendation*

The **Lightbox Effect** emphasizes that we are in a separate mode. As a consequence, it is not needed for most non-modal overlays. As an example, refer back to Figure 5-2, the Orbitz calendar pop up. Since the overlay is really more like an in-page widget, it would not be appropriate to make the chooser feel heavier by using a **Lightbox Effect**.

Tip

Use the **Lightbox Effect** to emphasize modality or call attention to special information in an overlay.

Staying in the flow

Overlays are a good way to avoid sending a user to a new page. This allows the user to stay within the context of the original page. However, since overlays are quick to display and inexpensive to produce, sometimes they can be tempting to use too freely, and in the process, may actually break the user's flow.

Anti-pattern: Idiot Boxes

Alan Cooper states a simple principle:

Don't stop the proceedings with idiocy.[*]

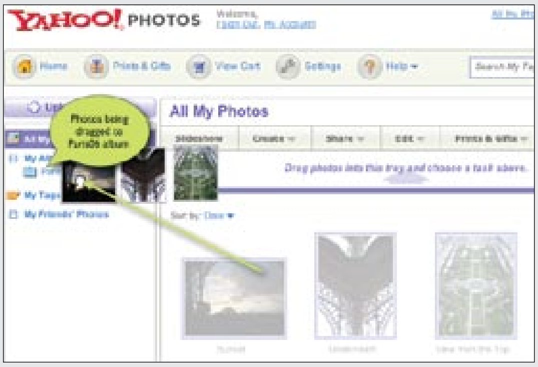
[*] Cooper, Alan and Robert Reimann. *About Face 2.0: The Essentials of Interaction Design* (Wiley, 1995), 178.

In the context of flow he describes how egregious it is to interrupt the user needlessly:

> *One form of excise is so prevalent that it deserves special attention.... Flow is a natural state, and people will enter it without much prodding. It takes some effort to break into flow after someone has achieved it. Interruptions like a ringing telephone will do it, as will an error message box. Some interruptions are unavoidable, but most others are easily dispensable.... Interrupting a user's flow for no good reason is* ***stopping the proceedings with idiocy*** *and is one of the most disruptive forms of excise.*[*]

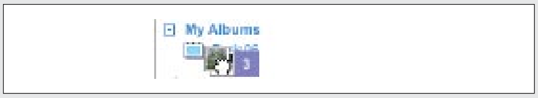
This is a clear anti-pattern that should be avoided. We call these types of overlays **Idiot Boxes**.

One of the clearest examples of **Idiot Boxes** is the way certain confirmation overlays were used in Yahoo! Photos. When users selected a set of photos and dragged and then dropped them onto an album, they were treated with not just one confirmation, but with two (Figure 5-6).


YAHOO! PHOTOS
All My Photos
Photos being dragged to Paris05 album
Drag photos into this tray and choose a task above.

Drag and drop

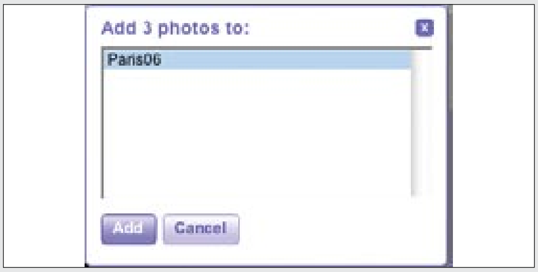
Photos are added to an album by drag and drop.



Album on drag over

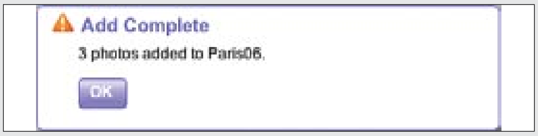
The album provides no visual feedback that it is a valid drop target.

* Cooper, Alan et al. *About Face 3: The Essentials of Interaction Design* (Wiley, 2007), 228.



Add Photos overlay

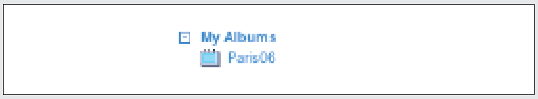
A confirmation overlay is posted that asks the users if they really want to drop the photos onto the album.



3 photos added to Paris06.

Add Complete overlay

Stating the obvious, the overlay repeats what just happened. Yes, the photos were added.



Album after drop

Notice that there is no difference between the album before or after the drop.

Figure 5-6. *Yahoo! Photos uses unnecessary overlay pop ups to communicate what happens when the user drops photos into an album*

A fundamental problem in this interaction is the lack of clear invitations and feedback at just the right moment. Lacking feedback when the user drags over the album (to signal you will be dropping it on the "Paris06" album) and after it has been dropped is often "remedied" by the heavy-handed use of pop-up overlays to state the obvious. We will discuss these concepts in more detail in later chapters.

Tip

Mind the interesting moments in an interaction. They can remove the need for **Idiot Boxes**.

Using JavaScript alerts

It is tempting to use the alert mechanism built into the browser for some confirmations. The problem with using this type of alerts is two-fold.

First, they do not present themselves consistently across different operating systems. Under Microsoft Windows they will appear centered in the browser window, but with the Macintosh they will slide out from under the title bar. Depending on where the action takes place, users may have to move their mouse a lot further each time they need to dismiss the alert (Figure 5-7).

Second, there is no way to control the look and feel of the alert pop up. With lightweight overlays, any valid web interface can be created to interact with the user.

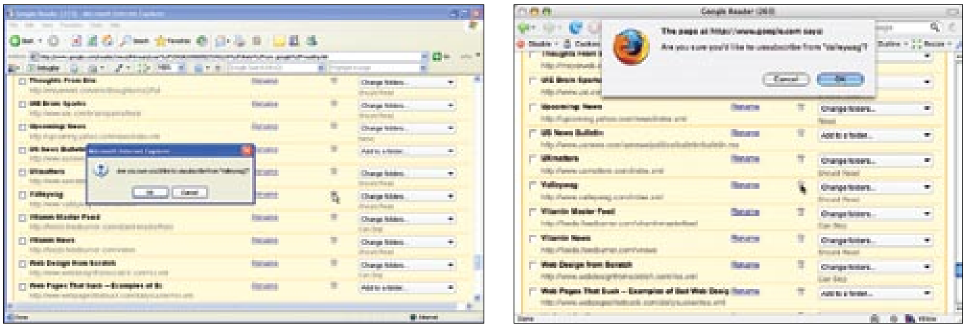


Figure 5-7. *In an older version of Google Reader, the JavaScript alert box displays differently on a Microsoft Windows PC versus a Macintosh*

The alert shown in Figure 5-7 has recently been replaced with a lightweight overlay (Figure 5-2).

Best Practices for Dialog Overlay

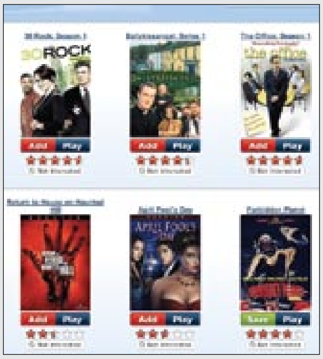
Here are some best practices to keep in mind:

- Always use **Dialog Overlays** instead of browser pop ups.
- Use the **Lightbox Effect** when the overlay contains important information, if the user should not ignore it, or if the interaction with the dialog should be modal.
- Avoid unnecessary **Dialog Overlays** (**Idiot Boxes**), as they interrupt the user's flow.
- Don't use an overlay when a simpler, in-page interaction would suffice (see discussion in the next chapter on **Inlays**).
- Avoid JavaScript alert boxes, as they don't provide a consistent user experience between operating systems.

Detail Overlay

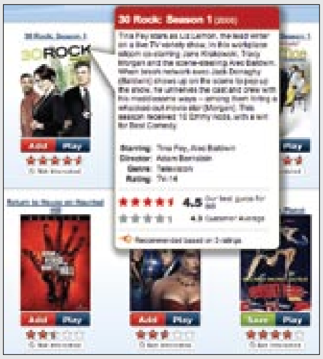
The second type of overlay is somewhat new to web applications. The **Detail Overlay** allows an overlay to present additional information when the user clicks or hovers over a link or section of content. Toolkits now make it easier to create overlays across different browsers and to request additional information from the server without refreshing the page.

Taking another example from Netflix, information about a specific movie is displayed as the user hovers over the movie's box shot (Figure 5-8).



Box shots

In the more recent versions of the Netflix site, large box shots are employed without synopsis text. Box shots convey a lot of information.

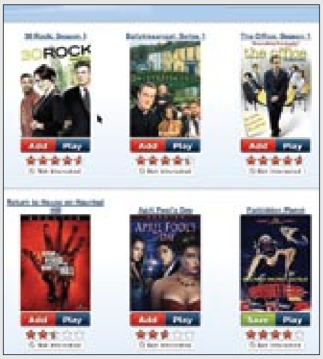


Detail overlay activation

However, often more information is needed to decide whether a movie should be played or added to a movie queue.

By providing a synopsis along with personalized recommendation information, the user can quickly make a determination.

The movie detail information is displayed after a slight delay.



Detail overlay deactivation

Moving the mouse outside the box shot immediately removes the movie detail information.

Figure 5-8. *Netflix shows "back of the box" information in an overlay as the user hovers over a movie's box shot*

Considerations

The overlay provides a nice way to reveal a synopsis for a movie. In a sense it is like flipping over the DVD box and reading what is on the back.

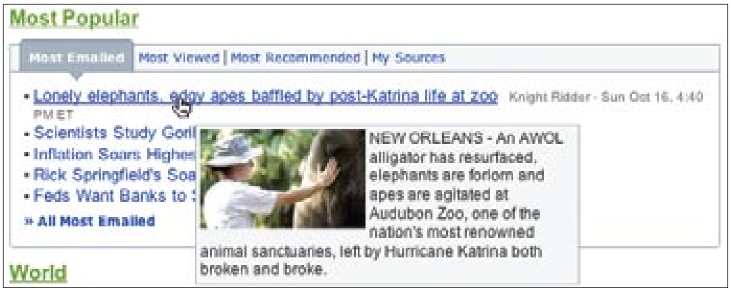
Activation

The overlay is displayed when the mouse hovers over a box shot. There is about a half-second delay after the user pauses over a movie. The delay on activation prevents users from accidentally activating the overlay as they move the cursor around the screen. Once the user moves the cursor outside the box shot, the overlay is removed immediately. Removing it quickly gives the user a fast way to dismiss it without having to look for a "Close" box.

Tip

For **Detail Overlays** activated by a mouse hover, provide a slight delay before displaying.

You can find the same interaction style employed on Yahoo! News. When the user hovers over news story links in various areas (like Most Popular), a sneak peek shows the news photo and lead of the story (Figure 5-9).



- Lonely elephants, edgy apes baffled by post-Katrina life at zoo Knight Ridder - Sun Oct 16, 4:40 PM ET
- Scientists Study Gori
- Inflation Soars Highes
- Rick Springfield's Soa
- Feds Want Banks to

NEW ORLEANS - An AWOL alligator has resurfaced, elephants are forlorn and apes are agitated at Audubon Zoo, one of the nation's most renowned animal sanctuaries, left by Hurricane Katrina both broken and broke.

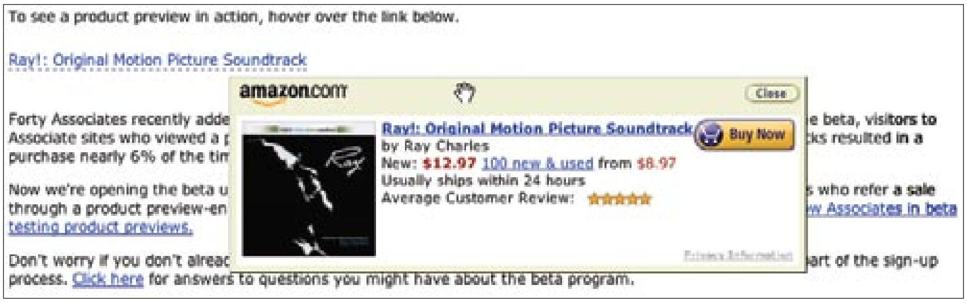
Figure 5-9. *Yahoo! News reveals more about a story without a user having to click to another page*

In both cases the user is given the additional context to make a decision about what to do next. In the case of Netflix, a movie's description or rating prediction may lead to the user renting the movie. With Yahoo! News, since click-throughs to news stories are more intentional, the user will be taken to stories that interest him. This creates a sense of user satisfaction and control, both of which are ingredients to a good user experience.

Anti-pattern: Mouse Traps

It is important to avoid activating the **Detail Overlay** too easily. We have seen usability studies that removed the delay in activation, and users reported that the interface was "too noisy" and "felt like a trap". We label this anti-pattern the **Mouse Trap.**

The reasoning for this is not clear, but Amazon uses the **Mouse Trap** anti-pattern in one of its "associate widgets". In Figure 5-10 the link "Ray! Original Motion Picture Soundtrack" activates an overlay providing information on the soundtrack and a purchase option. Presumably, this approach is intended to drive purchases—but it also presents an annoying experience.


To see a product preview in action, hover over the link below.

Ray!: Original Motion Picture Soundtrack

Forty Associates recently adde Associate sites who viewed a p purchase nearly 6% of the tin

Now we're opening the beta u through a product preview-en testing product previews.

Don't worry if you don't alreac process. Click here for answers to questions you might have about the beta program.

Figure 5-10. *Amazon shows a book-buying widget on simple hover—but requires clicking the "Close" box to dismiss it*

Anti-pattern: Non-Symmetrical Activation/Deactivation

When the user moves her mouse over the link, the overlay springs up immediately. The only way she can remove the overlay is by clicking the small close button in the upper right. Using **Non-Symmetrical Activation/Deactivation** is also a general anti-pattern that should be avoided. It should take the same amount of effort to dismiss an overlay as it took to open it.

Compare the Amazon approach to both the Netflix and Yahoo! News approaches. The activation is slightly harder (pause, slight delay) than the deactivation (immediate when mouse is moved away).

Tip

Make activation and deactivation symmetrical interactions.

Another example of **Non-Symmetrical Activation/Deactivation** turns up in a previous version of Yahoo! Foods (Figure 5-11). To see all main ingredients for a recipe, the user clicked a red arrow. This activated an overlay with the ingredients. However, clicking on the arrow again did not collapse the overlay. Instead, the user had to click on the close button (red X).

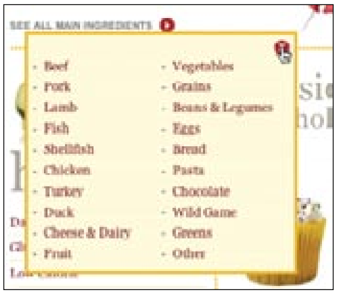


Figure 5-11. *Yahoo! Foods All Main Ingredients drop-down is activated by clicking on the arrow and can only be deactivated by clicking the close button (X)*

Anti-pattern: Needless Fanfare

One of the advantages of a lightweight overlay is the ability to pop it up quickly. After a slight delay in activation (recall the half-second delay used by Netflix), you would not want or need the overlay to come up slowly. But in the case of Borders online, this is precisely the approach taken (Figure 5-12). First the activation is immediate (no delay). This creates the noisy, mouse-trap interface just discussed. Second, there's a needless animation that zooms the box up into place and then back down when the mouse moves away from a book. **Needless Fanfare** is an anti-pattern to avoid.

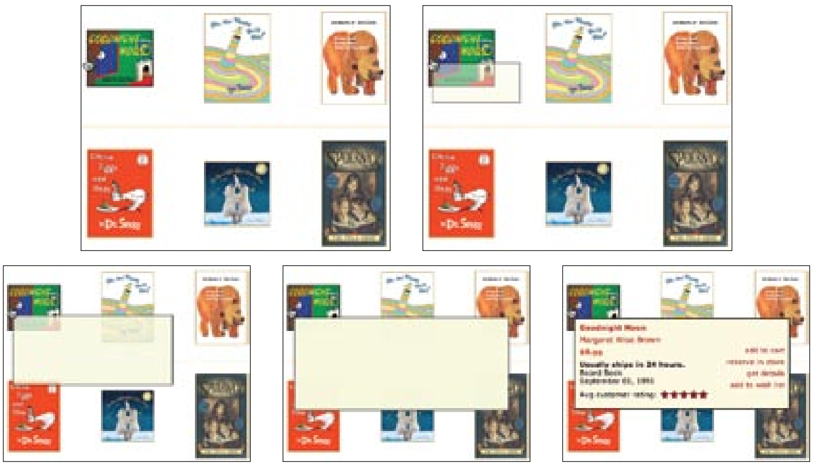


Figure 5-12. *Each Detail Overlay is preceded by the Needless Fanfare of a one-second animation that zooms the information into place*

The animation takes a full second to complete. Instead of delaying before activation, it delays after activation. Perhaps the design team thought that the animation tied the details to the item it zoomed out from. However, the Netflix approach simply creates a bubble effect that points back to the item it is revealing information for (Figure 5-7, right).

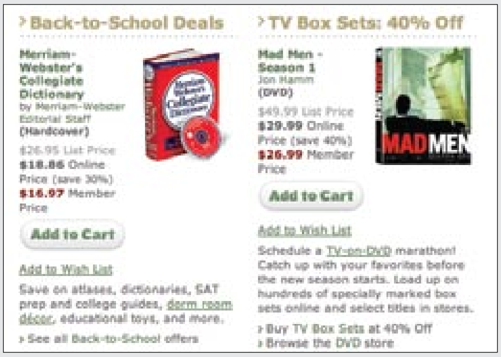
Tip

Once activated, show **Detail Overlays** immediately without fanfare.

Anti-pattern: Hover and Cover

We discussed the anti-pattern **Hover and Cover** in Chapter 2, and it's important to keep this anti-pattern in mind when providing a **Detail Overlay**. In the Netflix example (Figure 5-7), the overlay does not get in the way of moving to the next box shot. Even though it covers the neighboring box shot, moving the mouse outside the original one removes the overlay immediately, providing a clear path to get an overlay on the next box shot.

Compare this to the **Detail Overlay** provided by barnesandnoble.com (Figure 5-13).



› Back-to-School Deals

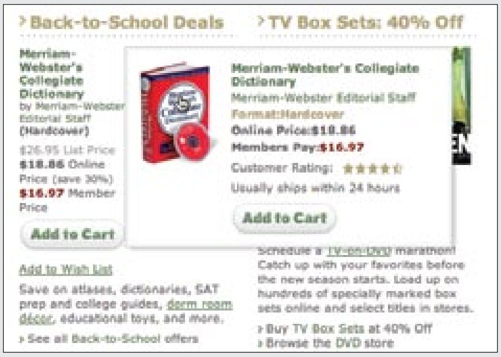
Merriam-Webster's Collegiate Dictionary
by Merriam-Webster Editorial Staff
(Hardcover)
$26.95 List Price
$18.86 Online Price (save 30%)
$16.97 Member Price
Add to Cart
Add to Wish List
Save on atlases, dictionaries, SAT prep and college guides, dorm room décor, educational toys, and more.
› See all Back-to-School offers

› TV Box Sets: 40% Off

Mad Men - Season 1
Jon Hamm
(DVD)
$49.99 List Price
$29.99 Online Price (save 40%)
$26.99 Member Price
Add to Cart
Add to Wish List
Schedule a TV-on-DVD marathon! Catch up with your favorites before the new season starts. Load up on hundreds of specially marked box sets online and select titles in stores.
› Buy TV Box Sets at 40% Off
› Browse the DVD store

Product display

On the Barnes & Noble home page, books are displayed with lots of detailed information.



› Back-to-School Deals

Merriam-Webster's Collegiate Dictionary
by Merriam-Webster Editorial Staff
(Hardcover)
$26.95 List Price
$18.86 Online Price (save 30%)
$16.97 Member Price
Add to Cart
Add to Wish List
Save on atlases, dictionaries, SAT prep and college guides, dorm room décor, educational toys, and more.
› See all Back-to-School offers

› TV Box Sets: 40% Off

Merriam-Webster's Collegiate Dictionary
Merriam-Webster Editorial Staff
Format:Hardcover
Online Price:$18.86
Members Pay:$16.97
Customer Rating: ★★★★½
Usually ships within 24 hours
Add to Cart

Schedule a TV-on-DVD marathon! Catch up with your favorites before the new season starts. Load up on hundreds of specially marked box sets online and select titles in stores.
› Buy TV Box Sets at 40% Off
› Browse the DVD store

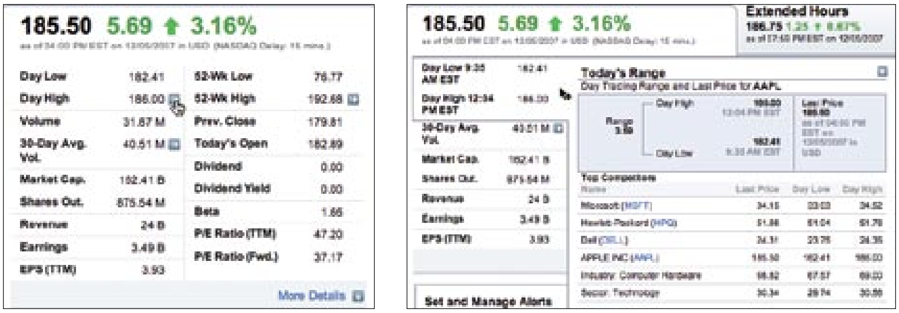
Detail Overlay

The **Detail Overlay** is an exact copy of the information displayed on the page. Also notice that the overlay covers most of the other item (*Mad Men*).

Figure 5-13. *Barnes & Noble does not need Detail Overlay since the information is exactly the same as displayed on the page*

Barnes & Noble uses a **Detail Overlay** in a completely useless manner. The overlay contains exactly the same information that the page already contained! Not only that, but the overlay almost completely obscures the other item displayed on the page. It also creates another common problem—the book image (dictionary) in the overlay is not positioned the same as the book image on the page (about 12 pixels difference). The shift causes the annoying illusion that the book is wiggling back and forth, which detracts from the experience.

On the other hand, AOL Finance provides a very nice example of pulling detailed stock information into an overlay that fits in a seamless manner, provides additional information, and does not cover up the navigation to the next item that might be of interest (Figure 5-14).



185.50 5.69 3.16%

Day Low	182.41	52-Wk Low	76.77
Day High	186.00	52-Wk High	192.68
Volume	31.67 M	Prev. Close	179.81
30-Day Avg. Vol.	40.51 M	Today's Open	182.89
		Dividend	0.00
Market Cap.	162.41 B	Dividend Yield	0.00
Shares Out.	875.54 M	Beta	1.66
Revenue	24 B	P/E Ratio (TTM)	47.20
Earnings	3.49 B	P/E Ratio (Fwd.)	37.17
EPS (TTM)	3.93		

More Details

185.50 5.69 3.16%

Extended Hours

Day Low 9:35 AM EST	182.41
Day High 12:34 PM EST	186.00
30-Day Avg. Vol.	40.51 M
Market Cap.	162.41 B
Shares Out.	875.54 M
Revenue	24 B
Earnings	3.49 B
EPS (TTM)	3.93

Set and Manage Alerts

Today's Range

Day Trading Range and Last Price for AAPL

Day High 186.00

Day Low 182.41

Last Price

Top Competitors

Name	Last Price	Day Low	Day High
Microsoft (MSFT)	34.15	[illegible]	34.52
Hewlett-Packard (HPQ)	51.86	51.04	51.78
Dell (DELL)	24.31	23.75	24.35
APPLE INC (AAPL)	185.50	182.41	186.00
Industry: Computer Hardware	[illegible]	[illegible]	[illegible]
Sector: Technology	30.34	29.74	30.50

Figure 5-14. *AOL Finance provides additional information without preventing other navigation; the detail is activated by clicking on the blue arrow button*

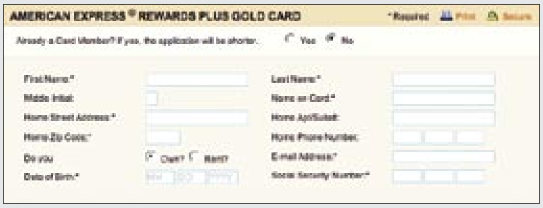
Best Practices for Detail Overlay

Here are some best practices to keep in mind:

- Use **Detail Overlays** to give a sneak peek at detailed information. This will avoid unnecessary page transitions.
- For hover-activated **Detail Overlays**, provide a slight delay for activation (about half a second). This will avoid the interface behaving like a **Mouse Trap**.
- For hover-activated **Detail Overlays**, provide a simple deactivation (e.g., simple mouse out).
- For **Detail Overlays**, make activation and deactivation symmetrical (don't make it harder to get out of the overlay than it was to activate it).
- Once activated, avoid lengthy animations or effects that delay showing a **Detail Overlay**. Use hover when clicking the object is reserved for a different action. In Yahoo! News, clicking on the link takes you to the news story page. Hovering gives you a sneak peek.
- Use hover when it is not obvious how to get more information. If you use hover to activate the details, you don't need additional user interface controls for activation.
- Use click if you prefer to make getting details more explicit. Usually you will couple this with a "see more" or a button that exposes more detail. Yahoo! Foods (Figure 5-11) uses a red arrow to indicate drop-down information. AOL Finance uses a blue arrow button to indicate more information (Figure 3-10).
- Use click if you will be providing additional links inside the overlay. Trying to keep an overlay in place (that was activated by hover) and click a link inside it is difficult.

Input Overlay

Input Overlay is a lightweight overlay that brings additional input information for each field tabbed into. American Express uses this technique in its registration for premium cards such as its gold card (Figure 5-15).

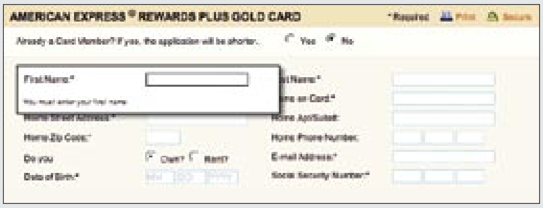

AMERICAN EXPRESS® REWARDS PLUS GOLD CARD *Required Print Secure

Already a Card Member? If yes, the application will be shorter. Yes [✓ No]

First Name:*		Last Name:*	
Middle Initial:		Name on Card:*	
Home Street Address:*		Home Apt/Suite#:	
Home Zip Code:*		Home Phone Number:	
Do you	[✓ Own?] Rent?	E-mail Address:*	
Date of Birth:*	MM DD YYYY	Social Security Number:*	

Input form

The form is displayed with simple prompt/input for each field. No additional help information is shown statically.


AMERICAN EXPRESS® REWARDS PLUS GOLD CARD *Required Print Secure

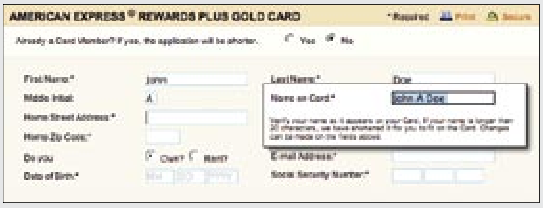
Already a Card Member? If yes, the application will be shorter. Yes [✓ No]

First Name:*

You must enter your first name.

Input overlay

Tabbing or clicking into any field wraps the field in an overlay. The overlay provides additional input information.


AMERICAN EXPRESS® REWARDS PLUS GOLD CARD *Required Print Secure

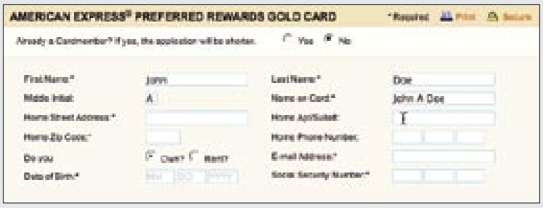
Already a Card Member? If yes, the application will be shorter. Yes [✓ No]

First Name:*	John	Last Name:*	Doe
Middle Initial:	A	Name on Card:*	John A Doe
Home Street Address:*			
Home Zip Code:*			
Do you	[✓ Own?] Rent?	E-mail Address:*	
Date of Birth:*	MM DD YYYY	Social Security Number:*	

Verify your name as it appears on your Card. If your name is longer than 20 characters, we have shortened it for you to fit on the Card. Changes can be made on the fields above.

Obscuring fields

The overlay does obscure fields just below it, but not to the left or right.


AMERICAN EXPRESS® PREFERRED REWARDS GOLD CARD *Required Print Secure

Already a Cardmember? If yes, the application will be shorter. Yes [✓ No]

First Name:*	John	Last Name:*	Doe
Middle Initial:	A	Name on Card:*	John A Doe
Home Street Address:*		Home Apt/Suite#:	
Home Zip Code:*		Home Phone Number:	
Do you	[✓ Own?] Rent?	E-mail Address:*	
Date of Birth:*	MM DD YYYY	Social Security Number:*	

Deactivation

Clicking anywhere removes the overlay. This lets the user click through the field covered by the overlay.

Figure 5-15. *American Express provides Input Overlays to guide the user through the signup process*

Considerations

There are a few things to keep in mind when using **Input Overlays**.

Clear focus

As the user tabs or clicks from field to field, the field gets wrapped in an overlay. The overlay contains additional input help information. This allows the normal display of the form to be displayed in a visually simple manner (just prompts and inputs). The overlay creates focus on the given input field. Instead of seeing an ocean of inputs, the user is focused on just entering one field.

Display versus editing

Additionally, when the **Input Overlay** is shown, the prompt is displayed in exactly the same manner as when the overlay doesn't show. This is critical, as it makes the overlay feel even more lightweight. If the overlay prompt were bold, for example, the change would be slightly distracting and take the focus away from input. The only difference between the non-overlay field and the overlay version is a slightly thicker input field border. This draws the eye to the task at hand—input.

Anti-pattern: Hover and Cover

But what about the anti-pattern, **Hover and Cover**? Doesn't this cause the same issues that we saw earlier? For example, in Figure 5-15 ("Obscuring fields"), the "Name on Card" overlay hides the "Home Apt/Suite#" and "Home Phone Number Fields" fields below it. There are several reasons that American Express was able to employ this overlay in forms and "get away" with covering some fields during input:

Field traversal
: The field traversal is side-to-side. The overlay for the "First Name" field (Figure 5-15, "Input overlay") does not obscure the next field, "Last Name".

Tab navigation
: Since tabbing is a primary navigation for forms, the mouse is not needed to navigate. This allows navigation to fields even if they were covered.

One-click deactivation
: Clicking anywhere hides the overlay. That means that when the "Name on Card" overlay is shown (Figure 5-15, "Obscuring fields"), clicking anywhere in the "Home/Apt Suite#" field will remove the overlay, allowing the user to click in the previously hidden field (Figure 5-15, "Deactivation").

Two additional touches that would help with field-covering issues include:

- Give the overlay a slight translucency in order to faintly reveal the fields below.
- Allow a click deactivation to not only deactivate the overlay but click through to the field that was clicked over. This allows the user to click into a field shown through the overlay.

Best Practices for Input Overlay

Here are some best practices to keep in mind:

- Use **Input Overlays** to simplify the visual style of a form. Place additional help in the overlay.
- For **Input Overlays**, make sure the only visual change between the field and the overlay field is intentional (e.g., making input field visually bolder).
- For **Input Overlays**, allow clicking anywhere to deactivate the overlay.

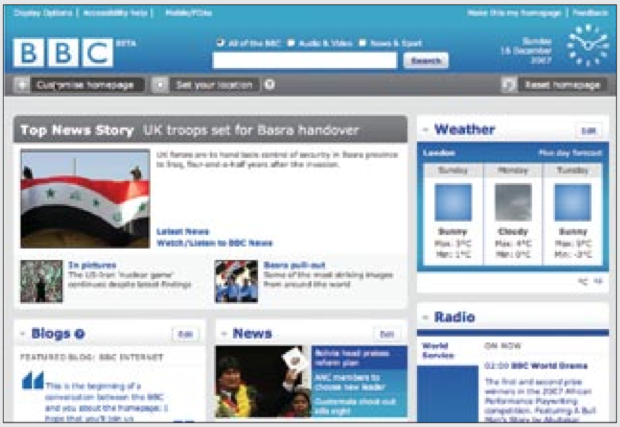
Chapter 6

Inlays

Not every bit of additional control, information, or dialog with the user needs to be an overlay. Another approach is to inlay the information directly within the page itself. To distinguish from the pop-up overlay, we call these in-page panels **Inlays**.

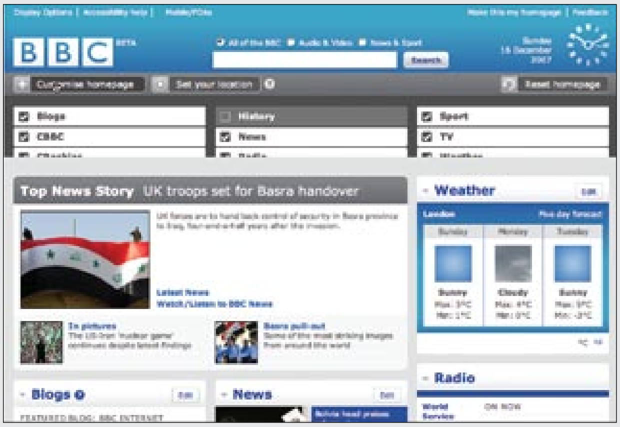
Dialog Inlay

A simple technique is to expand a part of the page, revealing a dialog area within the page. The BBC recently began experimenting with using a **Dialog Inlay** as a way to reveal customization controls for its home page (Figure 6-1).



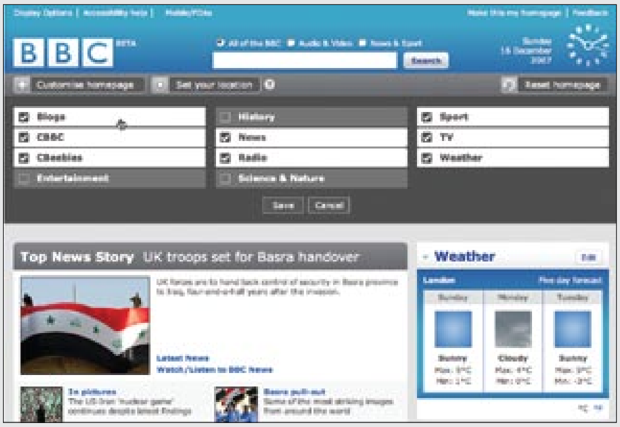
Activation

The "Customize homepage" button activates the customization inlay.



Customization inlay (slide)

The inlay slides into view.



Customization inlay

The additional customization controls for the BBC home page are shown directly in context with the rest of the page.

Figure 6-1. *The BBC home page puts its customization tools in an inlay that slides out when activated*

Considerations

Of course an overlay could have been used instead. However, the problem with overlays is that no matter where they get placed, they will end up hiding information. Inlays get around this problem by inserting themselves directly into the context of the page.

Tip

Inlays provide in-context dialog with the user.

In context

This **Dialog Inlay** is similar to a drawer opening with a tray of tools. Instead of being taken to a separate page to customize the home page appearance, the user can make changes and view the effects directly. The advantage is the ability to tweak the page while viewing the actual page.

My Yahoo! also provides a **Dialog Inlay** for revealing its home page customization tools. The original version of My Yahoo! did not use this approach. Instead, customizations would take the user away to a separate page (Figure 6-2).

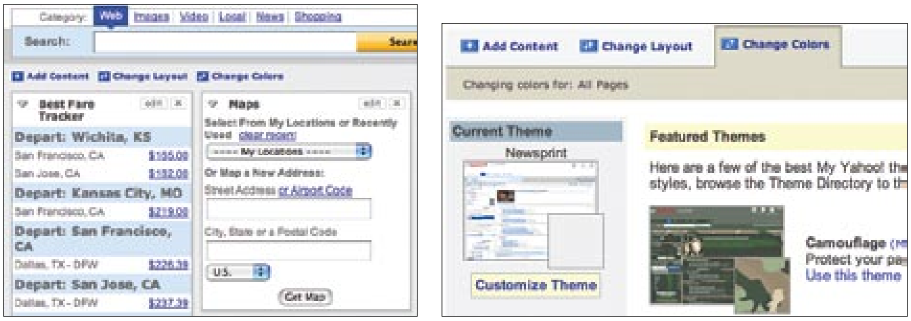

Category: Web Images Video Local News Shopping
Search:
Add Content
Change Layout
Change Colors
Best Fare Tracker
Depart: Wichita, KS
San Francisco, CA $155.00
San Jose, CA $152.00
Depart: Kansas City, MO
San Francisco, CA $219.00
Depart: San Francisco, CA
Dallas, TX - DFW $226.39
Depart: San Jose, CA
Dallas, TX - DFW $237.39
Maps
Select From My Locations or Recently Used clear recent
---- My Locations ----
Or Map a New Address:
Street Address or Airport Code
City, State or a Postal Code
U.S.
Get Map
Changing colors for: All Pages
Current Theme
Newsprint
Customize Theme
Featured Themes
Here are a few of the best My Yahoo! the styles, browse the Theme Directory to th
Camouflage
Protect your pa
Use this theme

Figure 6-2. *In an original version of My Yahoo!, clicking on "Change Colors" takes you to a separate page to customize colors*

Figure 6-3 shows the new My Yahoo!, incorporating an **Dialog Inlay** that slides into place to reveal customization tools for changing the page's appearance.

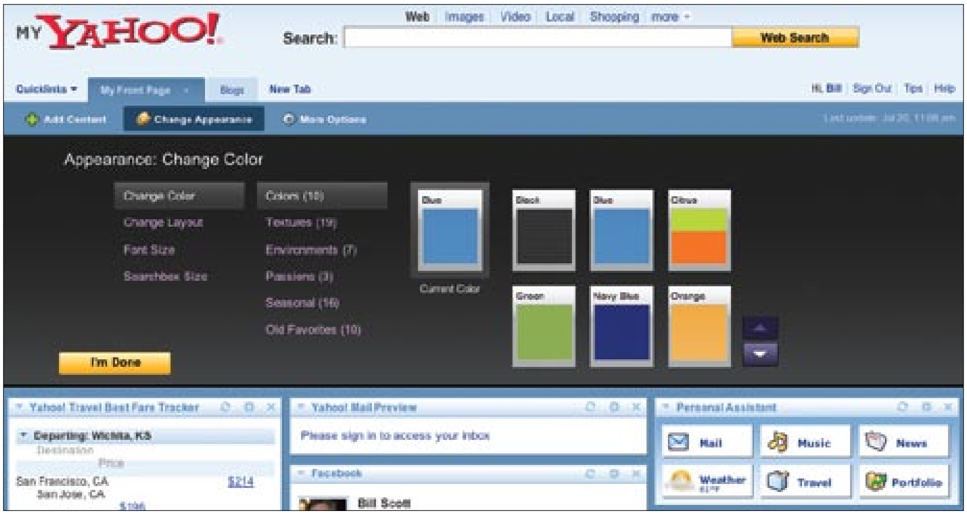
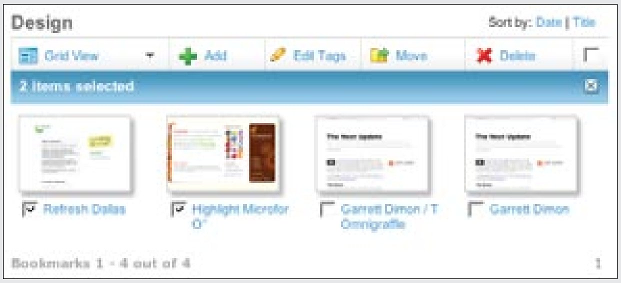


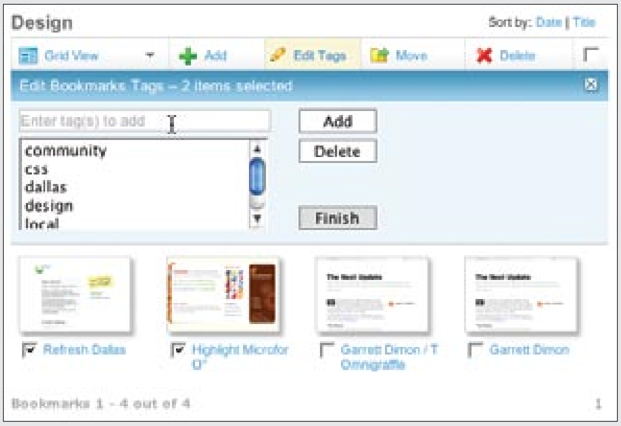
Figure 6-3. *My Yahoo! allows the page to be customized inline by using an Inlay Dialog panel that slides into view*

Here's one more example, which shows how a **Dialog Inlay** can be used to operate on specific objects within an interface. In Yahoo! Bookmarks, selected bookmarks can be edited, deleted, etc. Because a pop up will often hide the items being operated on (the user ends up having to move the dialog pop up out of the way to make sure the right object is being deleted), an inlay can work nicely when tied in with a toolbar (Figure 6-4).



Toolbar + selected items

Two bookmarks are selected. The toolbar at the top operates on the selected bookmarks.



Edit tags

Clicking the "Edit Tags" button slides an "Edit Bookmark Tags" panel into place. The panel provides full editing for the selected bookmarks. The bookmarks gallery is still visible.

Figure 6-4. *In Yahoo! Bookmarks, clicking "Edit Tags" expands a Dialog Inlay for editing the two bookmarks selected*

By being in proximity to the toolbar and keeping the objects visible, the **Dialog Inlay** provides a clear way to expose additional interface elements in context. In all of the previous examples, the panel is revealed with an animated slide in transition. This provides a nice way to smooth out the experience (we will discuss transitions in Chapters 11 and 12).

Best Practices for Dialog Inlay

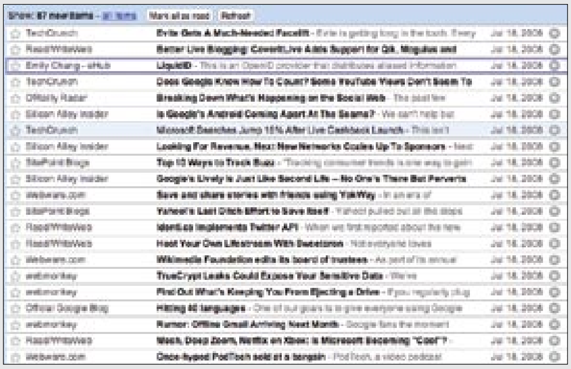
Here are some best practices to keep in mind:

- Use **Dialog Inlays** for page customization. It is helpful to tweak the page and see the results at the same time.
- To smooth the introduction of the **Dialog Inlay** into the page, use a quick slide in animation.
- Use **Dialog Inlays** to connect the dialog with the element it slides out from.
- Use **Dialog Inlays** for secondary tools that aren't primary to the main flow of the page.

List Inlay

Lists are a great place to use **Inlays**. Instead of requiring the user to navigate to a new page for an item's detail or popping up the information in an **Overlay**, the information can be shown with a **List Inlay** in context. The **List Inlay** works as an effective way to hide detail until needed—while at the same time preserving space on the page for high-level overview information.

Google Reader provides an expanded view and a list view for unread blog articles. In the list view, an individual article can be expanded in place as a **List Inlay** (Figure 6-5).

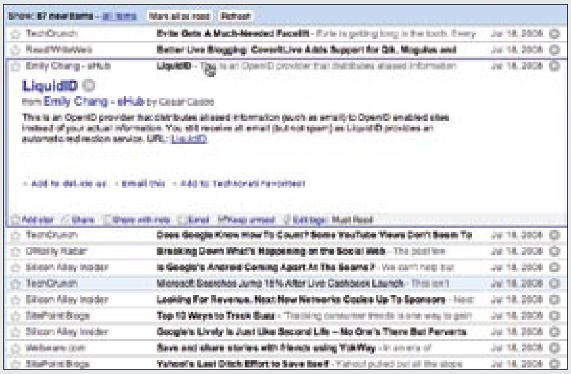


Show: 67 new items - all items | Mark all as read | Refresh

TechCrunch	**Evite Gets A Much-Needed Facelift** - Evite is getting long in the tooth. Every	Jul 18, 2008
ReadWriteWeb	**Better Live Blogging: CoveritLive Adds Support for Qik, Mogulus and**	Jul 18, 2008
Emily Chang - eHub	**LiquidID** - This is an OpenID provider that distributes aliased information	Jul 18, 2008
TechCrunch	**Does Google Know How To Count? Some YouTube Views Don't Seem To**	Jul 18, 2008
O'Reilly Radar	**Breaking Down What's Happening on the Social Web** - The past few	Jul 18, 2008
Silicon Alley Insider	**Is Google's Android Coming Apart At The Seams?** - We can't help but	Jul 18, 2008
TechCrunch	Microsoft Searches Jump 15% After Live Cashback Launch - This isn't	Jul 18, 2008
Silicon Alley Insider	**Looking For Revenue, Hezz New Networks Cozies Up To Sponsors** - Next	Jul 18, 2008
SitePoint Blogs	**Top 10 Ways to Track Buzz** - "Tracking consumer trends is one way to gain	Jul 18, 2008
Silicon Alley Insider	**Google's Lively is Just Like Second Life -- No One's There But Perverts**	Jul 18, 2008
Webware.com	**Save and share stories with friends using YikWay** - In an era of	Jul 18, 2008
SitePoint Blogs	**Yahoo!'s Last Ditch Effort to Save Itself** - Yahoo! pulled out all the stops	Jul 18, 2008
ReadWriteWeb	**Identi.ca Implements Twitter API** - When we first reported about the new	Jul 18, 2008
ReadWriteWeb	**Host Your Own Lifestream With Sweetcron** - Not everyone loves	Jul 18, 2008
Webware.com	**Wikimedia Foundation edits its board of trustees** - As part of its annual	Jul 18, 2008
webmonkey	**TrueCrypt Leaks Could Expose Your Sensitive Data** - We've	Jul 18, 2008
webmonkey	**Find Out What's Keeping You From Ejecting a Drive** - If you regularly plug	Jul 18, 2008
Official Google Blog	**Hitting 40 languages** - One of our goals is to give everyone using Google	Jul 18, 2008
webmonkey	**Rumor: Offline Gmail Arriving Next Month** - Google fans the moment	Jul 18, 2008
ReadWriteWeb	**Week, Deep Zoom, Netflix on Xbox: Is Microsoft Becoming "Cool"?** -	Jul 18, 2008
Webware.com	**Once-hyped PodTech sold at a bargain** - PodTech, a video podcast	Jul 18, 2008

Google Reader list view

In list view, articles are shown as a list of blog article titles.


Show: 87 new items - all items | Mark all as read | Refresh

TechCrunch — Evite Gets A Much-Needed Facelift - Evite is getting long in the tooth. Every — Jul 18, 2008
ReadWriteWeb — Better Live Blogging: CoveritLive Adds Support for Qik, Mogulus and — Jul 18, 2008
Emily Chang - eHub — LiquidID - This is an OpenID provider that distributes aliased information — Jul 18, 2008

LiquidID
from Emily Chang - eHub by César Castro
This is an OpenID provider that distributes aliased information (such as email) to OpenID enabled sites instead of your actual information. You still receive all email (but not spam) as LiquidID provides an automatic redirection service. URL: LiquidID

- Add to del.icio.us - Email this - Add to Technorati Favorites!

Add star | Share | Share with note | Email | Keep unread | Edit tags: Must Read

TechCrunch — Does Google Know How To Count? Some YouTube Views Don't Seem To — Jul 18, 2008
Officially Radar — Breaking Down What's Happening on the Social Web - The past few — Jul 18, 2008
Silicon Alley Insider — Is Google's Android Coming Apart At The Seams? - We can't help but — Jul 18, 2008
TechCrunch — Microsoft Searches Jump 15% After Live Cashback Launch - This isn't — Jul 18, 2008
Silicon Alley Insider — Looking For Revenue, Ning: New Networks Cozies Up To Sponsors - Next — Jul 18, 2008
SitePoint Blogs — Top 10 Ways to Track Buzz - "Tracking consumer trends is one way to gain — Jul 18, 2008
Silicon Alley Insider — Google's Lively Is Just Like Second Life – No One's There But Perverts — Jul 18, 2008
Webware.com — Save and share stories with friends using YikWay - In an era of — Jul 18, 2008
SitePoint Blogs — Yahoo!'s Last Ditch Effort to Save Itself - Yahoo! pulled out all the stops — Jul 18, 2008

Inlay list

Clicking on a single article expands it in place, in context with the rest of the list.

Figure 6-5. *In list view, Google Reader shows all articles as a collapsed list—except for the one that is currently selected*

Considerations

By allowing the reader to move through a list of article titles (by mouse or keyboard), the articles can be scanned quickly so she can decide which should be looked at in detail. Clicking on an article title expands the article in place. Showing one item at a time focuses the reader on the current content.

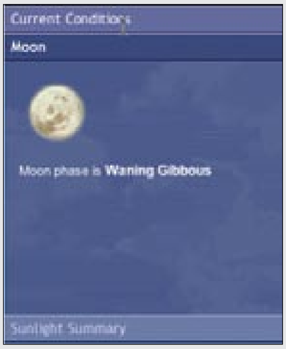
Tip

Expanding content in context is a powerful way to help users understand information. It follows the principle of focus+context.*

Accordion: One-at-a-time expand

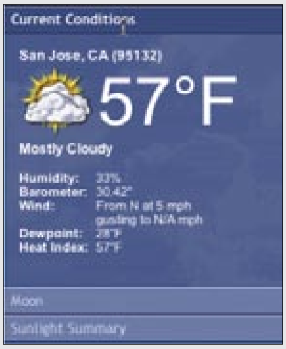
The **Accordion** is an interface element that employs the **List Inlay** pattern to show only one open panel in a list at a time. The following weather widget demo uses weather.com to display real-time weather information in an accordion (Figure 6-6).

* Focus+context enables users to see the object of interest in full detail while at the same time getting a overview of the surrounding context. It is also related to Jenifer Tidwell's design pattern, "Overview Plus Detail," described in her book *Designing Interfaces* (O'Reilly).



One panel at a time

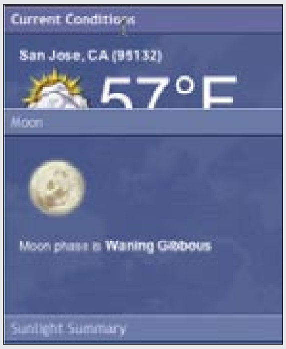
The weather demo uses an accordion to display a single panel of weather-related information at a time.



Opening and closing

Opening a new panel (“Current Conditions”) closes the previous panel (“Moon”).

The sliding transition makes it clear which panel is being switched to.



One panel at a time

It is important that the accordion control stay the same height throughout the sliding transition.

Figure 6-6. *The weather demo illustrates the way an accordion works: only one panel is revealed at a time*

In this example, the accordion allows only one weather pane to be visible at a time. When a pane is activated it slides open, simultaneously closing the previously open panel.

Accordions work best when:

- Each pane of content is independent
- Only one pane is visible at a time
- Each pane can be logically titled

Accordion: More than one pane visible at a time

You may want to allow multiple panes to be visible at a time if:

- Detail panes provide context for one another
- The vertical height of the accordion area is not fixed
- Different panels of content may be different heights

Parallel content

The Yahoo! Autos Car Finder tool (Figure 6-7) uses an accordion-style interaction for search filters that allows more than one pane to be open at a time. This choice makes sense because the decisions needed for one detail pane may be affected by the details of another pane. However, one problem with this specific implementation is the lack of information when a pane is closed. For example, no summary information is given for the "Price" tab. Looking at that button, it is not clear whether search criteria has been set or what it might be set to without first opening the pane.

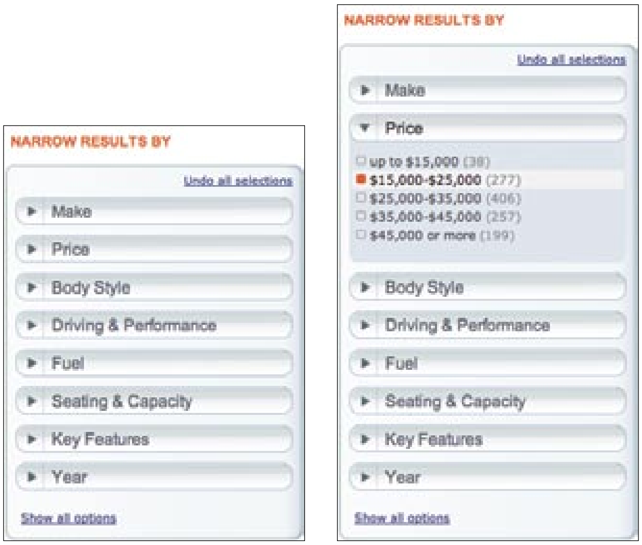
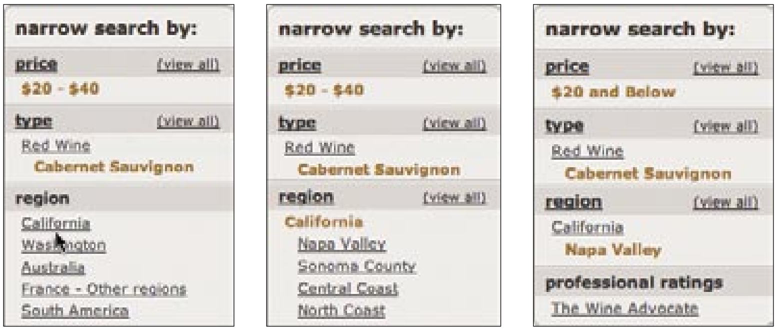


Figure 6-7. *Yahoo! Autos places filter criteria in Accordion panes; when panes are hidden, no summary information is provided*

Wine.com takes the opposite approach (Figure 6-8). Each area is initially open. All filters are displayed. As the user selects filters, the area collapses partially to show the results of filtering. This strategy accomplishes a few things. First, it gets rid of detail progressively—unnecessary information is hidden. Second, it invites the user to interact with the hidden content (with the "view all" link). Third, it provides summary information for panels that do not need to be opened as often.


narrow search by:
price (view all)
$20 - $40
type (view all)
Red Wine
Cabernet Sauvignon
region
California
Washington
Australia
France - Other regions
South America

narrow search by:
price (view all)
$20 - $40
type (view all)
Red Wine
Cabernet Sauvignon
region (view all)
California
Napa Valley
Sonoma County
Central Coast
North Coast

narrow search by:
price (view all)
$20 and Below
type (view all)
Red Wine
Cabernet Sauvignon
region (view all)
California
Napa Valley
professional ratings
The Wine Advocate

Figure 6-8. *Wine.com never closes a pane completely; as filters are selected, it partially closes and summarizes the choices*

Accordions can also be horizontally oriented. This is usually best done in nontraditional interfaces. A good example is the Xbox 360 interface (Figure 6-9).

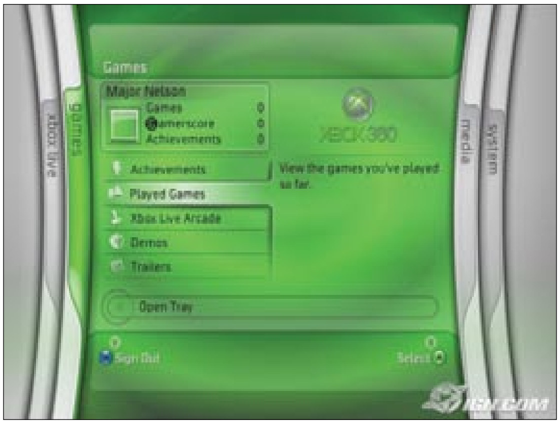


Figure 6-9. *The Xbox 360 provides a novel menu interface by using a horizontal accordion*

Even though the previous examples vary in visual and interaction styles, they all share the **List Inlay** pattern as a basic approach to revealing additional content in the context of the existing page.

Best Practices for List Inlay

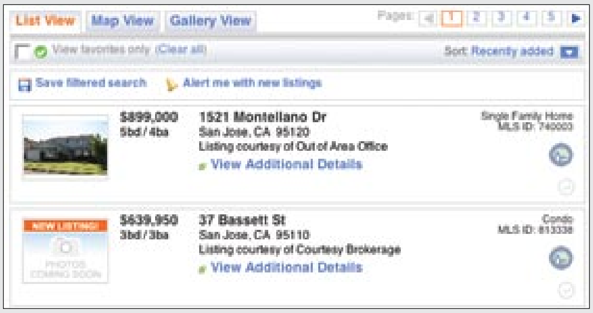
Here are some best practices to keep in mind:

- Use **List Inlay** when the context of the other items in the list help the user understand the visible inlay.
- Use **List Inlay** to avoid having users navigate to a new page or popping up an overlay to see an item's detail.
- Restrict it to show only a single item when you need to preserve space and the hidden content is not critical to the opened content.
- Allow multiple items to be visible for parallel content (like filters in a search).

Detail Inlay

A common idiom is to provide additional detail about items shown on a page. We saw this with the example of the Netflix movie detail pop up in Chapter 5 (Figure 5-8). Hovering over a movie revealed a **Detail Overlay** calling out the back-of-the-box information.

Details can be shown inline as well. Roost allows house photos to be viewed in-context for a real estate listing with a **Detail Inlay** (Figure 6-10).

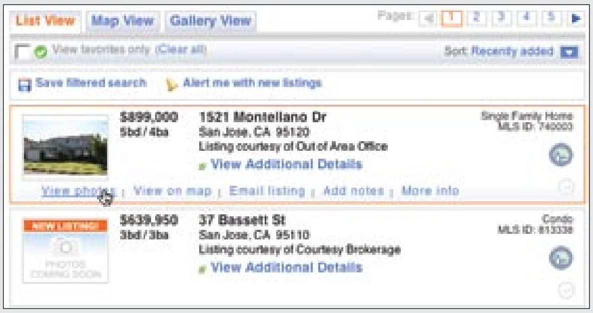


$899,000 5bd / 4ba
1521 Montellano Dr
San Jose, CA 95120
Listing courtesy of Out of Area Office
View Additional Details
Single Family Home
MLS ID: 740003

$639,950 3bd / 3ba
37 Bassett St
San Jose, CA 95110
Listing courtesy of Courtesy Brokerage
View Additional Details
Condo
MLS ID: 813338

In-context tools

Hovering over a real estate listing brings in a set of in-context tools, including the "View photos" tool.

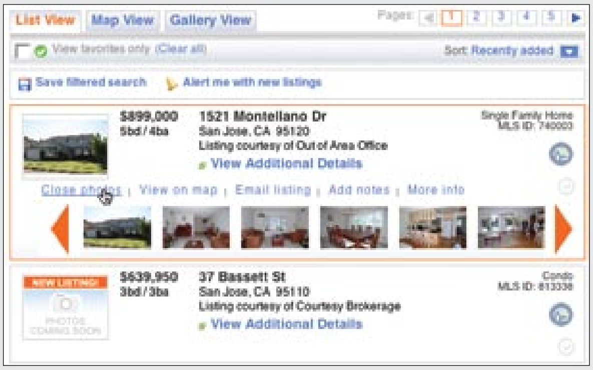

List View
Map View
Gallery View
Pages: 1 2 3 4 5
View favorites only (Clear all)
Sort: Recently added
Save filtered search
Alert me with new listings
View photos | View on map | Email listing | Add notes | More info
NEW LISTING!
PHOTOS COMING SOON

$899,000 5bd / 4ba
1521 Montellano Dr
San Jose, CA 95120
Listing courtesy of Out of Area Office
View Additional Details
Single Family Home
MLS ID: 740003

$639,950 3bd / 3ba
37 Bassett St
San Jose, CA 95110
Listing courtesy of Courtesy Brokerage
View Additional Details
Condo
MLS ID: 813338

House photos inlay

Clicking on the "View photos" link expands the real estate item to include a carousel of house photos.


List View
Map View
Gallery View
Pages: 1 2 3 4 5
View favorites only (Clear all)
Sort: Recently added
Save filtered search
Alert me with new listings
Close photos | View on map | Email listing | Add notes | More info
NEW LISTING!
PHOTOS COMING SOON

$899,000
5bd / 4ba
1521 Montellano Dr
San Jose, CA 95120
Listing courtesy of Out of Area Office
View Additional Details
Single Family Home
MLS ID: 740003

$639,950
3bd / 3ba
37 Bassett St
San Jose, CA 95110
Listing courtesy of Courtesy Brokerage
View Additional Details
Condo
MLS ID: 813338

Detail overlay

The **Detail Inlay** contains thumbnails of house photos. Clicking on an individual thumbnail pops up a **Detail Overlay** with a larger photo of the house.

Figure 6-10. *Roost provides both Detail Inlay and Detail Overlay patterns to show home photos*

Considerations

One of the more difficult things to do on most real estate sites is get a view of the house in context without having to navigate from page to page. The curb appeal, inside view, and backyard are all key factors in driving interest for a house. Knowing this, the team at Roost wanted to make it really easy to get to the photos quickly.

Combining inlays and overlays

Roost's solution was to combine several patterns. First, it uses the **Hover Reveal**, a **Contextual Tools** pattern, to reveal a set of tools when the user hovers over a listing. Second, it uses the **Detail Inlay** pattern to show a carousel of photos when the user clicks on the "View photos" link. And finally, it uses a **Detail Overlay** to blow up a thumbnail when clicked on.

Compare this to the traditional approach, one that requires the user to navigate from the listing page to a photo page and back again. The Roost team actually expended a Herculean effort in setting up this convenience, as it is dealing with hundreds of MLS listings with different contractual requirements for displaying real estate photos. The Roost team worked out the difficulties behind the scenes to create a nice user experience.

Best Practices for Detail Inlay

Here are some best practices to keep in mind:

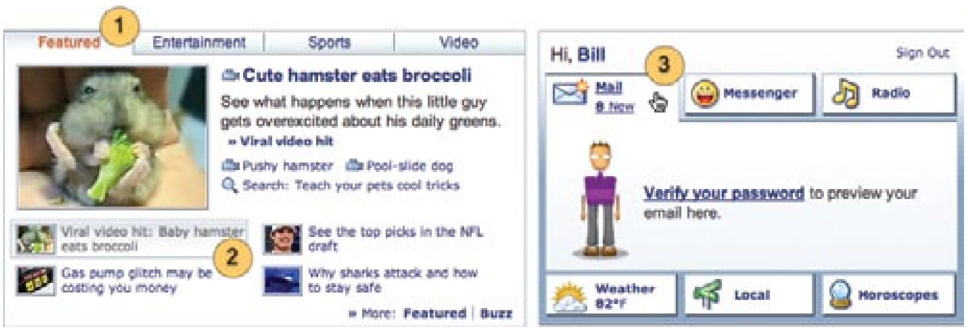
- Use **Detail Inlay** to provide additional information in context without hiding other information.
- Use **Detail Inlay** to avoid the anti-pattern **Hover and Cover**.
- Make it easy to dismiss the **Detail Inlay**.

Tabs

Lest we forget, there are some very traditional interface elements that can be used to inlay details. *Tabs*, for instance, can be used as a **Detail Inlay**. Instead of moving the user from page to page (site navigation), tabs can be used to bring in content within the page, keeping the user in the page.

Considerations

In Figure 6-11 you can see not just one type of tab for Yahoo!'s home page, but three! Three different tab styles and interactions can be confusing. In fact, user-testing revealed this to be the case. However, the design sidestepped some of the problem by giving the three tab areas each a different visual style. Let's look at each type of tab interaction.



Cute hamster eats broccoli

See what happens when this little guy gets overexcited about his daily greens.

» Viral video hit

Pushy hamster Pool-slide dog

Search: Teach your pets cool tricks

Viral video hit: Baby hamster eats broccoli

See the top picks in the NFL draft

Gas pump glitch may be costing you money

Why sharks attack and how to stay safe

» More: Featured | Buzz

Verify your password to preview your email here.

Figure 6-11. *Yahoo's home page employs three different Tab styles to encourage exploration of additional content*

Traditional tabs

The Featured, Entertainment, Sports, and Video sections are styled as a traditional tab notebook (Figure 6-11, callout ①). Going from section to section requires the user to click on the corresponding tab. Most studies on tabs find that the first and second tabs get the most activity and the subsequent tabs get clicked less frequently. This is the rationale for placing the "Featured" content first.

Content tabs

Each section has four featured stories. In this example, the “Cute hamster” is tied to the thumbnail and story lead (Figure 6-11, callout ②). Clicking on a story lead switches the content inside the tab. Effectively this is a secondary tab control, but visually it appears as a content story. By switching in the story without leaving the page, the user can get a peek at the top stories.

Personal assistant tabs

On the right side of the page, Yahoo! provides what it calls a *Personal Assistant*. Each tab in this area (Mail, Messenger, etc.) is activated by hovering over the tab. In our example the mouse is hovered over the Mail tab (Figure 6-11, callout ③) and it automatically expands open. Clicking on the link actually takes the user to Yahoo! Mail.

The three types of tabs vary greatly, visually and interactively. However, Yahoo! is able to pull this off because:

- Normal users of Yahoo! will discover these interactions over time.
- Creating the contrast makes a more visually compelling interface, as well as making the interaction feel deeper (inviting exploration).
- It is a great improvement over the old Yahoo! home page, which was completely static. Every link took the user to a different page. Keeping users on the page until they are ready to leave actually creates a happier user experience.

Best Practices for Tabs

Here are some best practices to keep in mind:

- Use tabs to display additional content inline.
- Avoid using multiple tabs on a single page.
- If you do use more than one set of tabs on a page, create a visual contrast to distinguish the tab content areas.
- Put the most important content in the first tab. Many users may not navigate to any of the other tabs.
- Activate tabs with mouse click.
- If revealing the content on other tabs is important, you can activate tabs on hover—but use this sparingly as it can be annoying (e.g., personal assistant tab on the Yahoo! home page).

Inlay Versus Overlay?

In these last two chapters we discussed overlays and inlays. How do we choose which makes sense in our interface? The following guidelines provide some suggestions on which to choose in a given situation:

- Use an overlay when there may be more than one place a dialog can be activated from (the exception may be showing details for items in a list).
- Use an overlay to interrupt the process.
- Use an overlay if there is a multi-step process.
- Use an inlay when you are trying to avoid covering information on the page needed in the dialog.
- Use an inlay for contextual information or details about one of many items (as in a list): a typical example is expanding list items to show detail.

Chapter 7
Virtual Pages

Overlays allow you to bring additional interactions or content in a layer above the current page. Inlays allow you to do this within the page itself. However, another powerful approach to keeping users engaged on the current page is to create a *virtual page*. That is to say, we create the illusion of a larger virtual page.

Tip

Think of the interface as a children's pop-up book. Additional real estate can be called on as needed to extend the interface.

Patterns that support virtual pages include:

- **Virtual Scrolling**
- **Inline Paging**
- **Scrolled Paging**
- **Panning**
- **Zoomable User Interface**

Virtual Scrolling

The traditional Web is defined by the "page." In practically every implementation of websites (for about the first 10 years of the Web's existence) pagination was the key way to get to additional content. Of course, websites could preload data and allow the user to scroll through it. However, this process led to long delays in loading the page. So most sites kept it simple: go fetch 10 items and display them as a page and let the user request the next page of content. Each fetch resulted in a page refresh.

The classic example of this is Google Search. Each page shows 10 results. Moving through the content uses the now-famous Google pagination control (Figure 7-1).

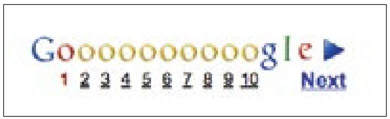
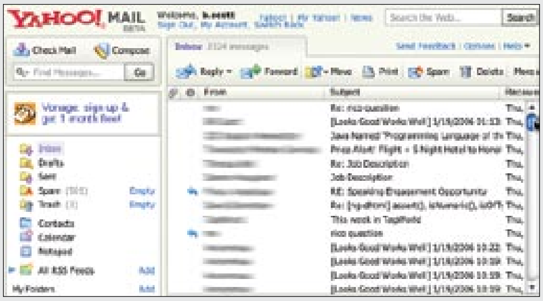


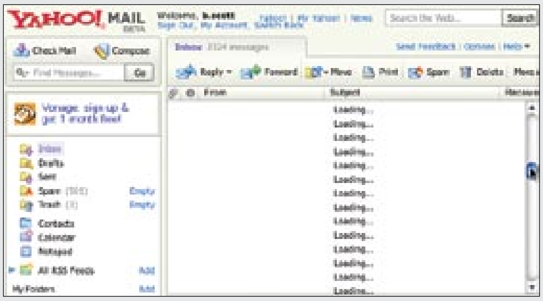
Figure 7-1. *The now-famous Google pagination control illustrates the most common way to move through data on the Web*

Another approach is to remove the artificial page boundaries created by paginating the data with **Virtual Scrolling.** In Yahoo! Mail, mail messages are displayed in a scrolled list that loads additional messages on demand as the user scrolls (Figure 7-2).



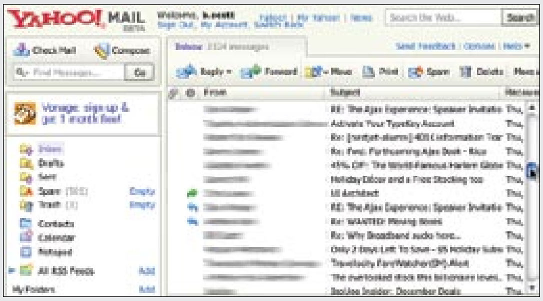
Scrolled list

Email messages are displayed as a scrolled list. This has been the normal approach on desktop mail clients. Yahoo! Mail brings that approach to the Web.



Scrolling

Messages are loaded on demand. As the user scrolls, the content items are filled in. While loading, the message lines are replaced with the text "Loading...".



Scroll completes

Messages are displayed based on where the user scrolled to.

Figure 7-2. *Instead of showing just the messages that can be displayed on a single page, Yahoo! Mail dynamically loads messages as the user scrolls*

Considerations

In some sense **Virtual Scrolling** turns the scrollbar into a "pagination control." But instead of a page refresh, the messages are seamlessly brought into the message-list pane, giving the illusion of a larger virtual space.

Desktop-style applications

In testing **Virtual Scrolling**, Yahoo! found that users naturally understood the scrolling paradigm, most likely because they were already accustomed to this feature on desktop mail clients. Since the Yahoo! Mail Web application looks very similar to desktop web applications, the expectation for scrolling already exists in the user's mind.

Tip

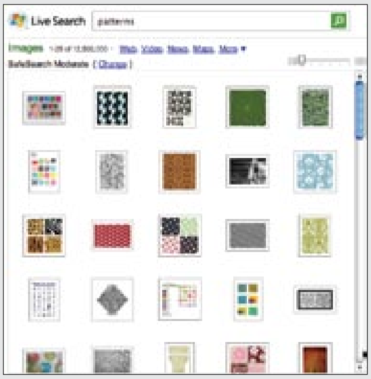
The more the web application looks and behaves like a desktop application, the more intuitive desktop idioms (like **Virtual Scrolling**) are to the user.

Loading status

There are a few downsides to the Yahoo! Mail version of **Virtual Scrolling**. First, if the loading is slow, it spoils the illusion that the data is continuous. Second, since the scrollbar does not give any indication of where users are located in the data, they have to guess how far down to scroll. A remedy would be to apply a constantly updating status while the user is scrolling.

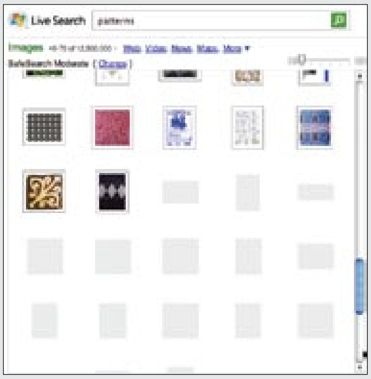
Progressive loading

Microsoft has applied **Virtual Scrolling** to its image search. However, it implements it in a different manner than Yahoo! Mail. Instead of all content being virtually loaded (and the scrollbar reflecting this), the scrollbar reflects what has been loaded. Scrolling to the bottom causes more content to load into the page (Figure 7-3).



Scrolled list

12,500,000 image results are represented as a scrolled list. Obviously there is no way to accurately represent that many items in a list with a scrollbar. Notice the scrollbar shows size relative to the amount of data that has been loaded.

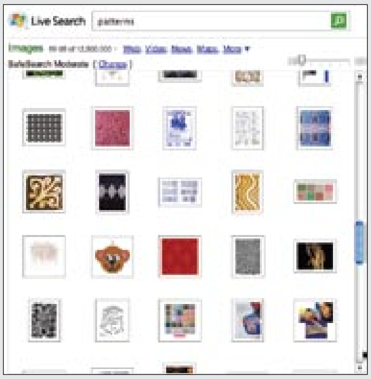


Scrolling

By scrolling into the area where results have not been loaded, images are initially represented as gray squares to indicate that they are currently not loaded.

As each image is loaded it replaces the gray squares.

At the top, the start and end range of the visible images is displayed ("Images 46–70 of 12,500,00").



Scroll completes

Image results are fully loaded, and the scrollbar is updated to reflect where this page is in relation to the previously loaded content.

Figure 7-3. *Microsoft Live Image Search uses Virtual Scrolling to fetch additional search results*

The Live Image Search approach does a nice job of eliminating paging. It gets around the "Where am I?" issue by progressively loading and dynamically adapting the scrollbar to reflect how much has been loaded.

Tip

Use progressive loading of content when the data size is large or unknown.

This type of **Virtual Scrolling** (where the scrollbar only reflects what has been directly loaded) works well for search results since relevance starts dropping off the further you move through the data. But with mail messages, this is not the case and would not be a good approach since users need to access messages beyond just those loaded at the top. The Microsoft approach also provides no way to jump to the bottom (although in this case the option is not needed since the end of all results has little or no relevance).

One more example illustrates an endless wall of pictures and uses a novel approach to a scrollbar control for **Virtual Scrolling**. PicLens is a Firefox add-on that allows viewing images from Google Search, Flickr, and other services to be displayed in the browser as a wall of photos that can be scrolled through (Figure 7-4).

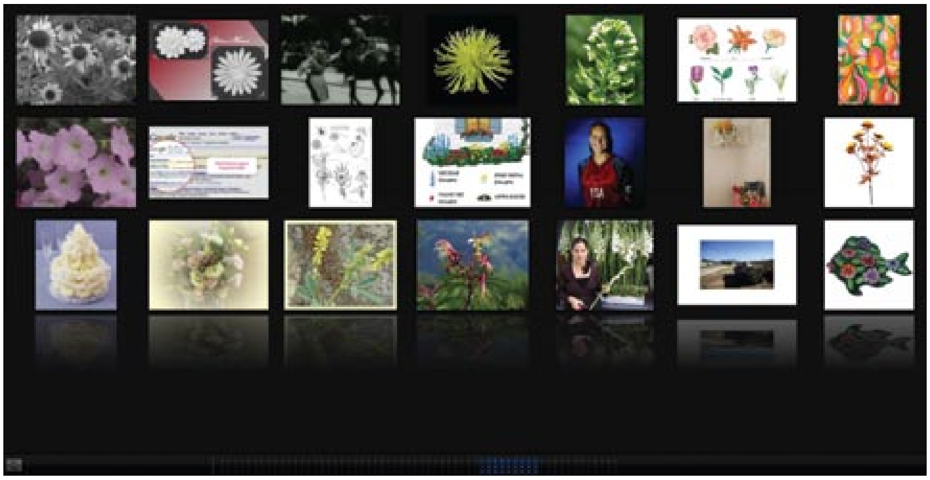


Figure 7-4. *PicLens provides an endless wall of photos; the scroller at the bottom continues to expand with more content*

The scrollbar dynamically resizes and relocates its thumb based on the amount of content loaded and where it is in the data (Figure 7-5). It also cleverly discards the beginning as it gets further and further into the content. The basic idea is to give users good navigation no matter where they are in the gallery of images.

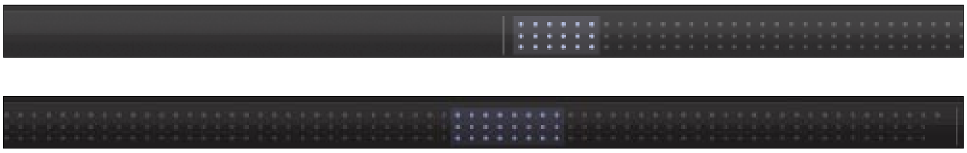


Figure 7-5. *The scroller in PicLens dynamically adapts to show the user's orientation in the data; it indicates how many images are shown and to some degree where the user is in the wall of images*

These examples of **Virtual Scrolling** demonstrate three different ways to manage the virtual space:

- Yahoo! Mail creates the illusion that all data has been loaded up-front by having the scrollbar reflect the total virtual space.
- Microsoft Live Search creates the virtual illusion as the user moves down through the search results.

- And PicLens does the same with the caveat that it shows a virtual window in the larger virtual space (by only providing a scroller control for where the user is and some before and after context).

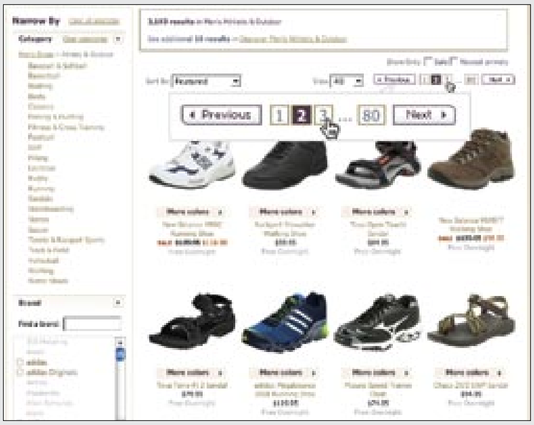
Best Practices for Virtual Scrolling

Here are some best practices to keep in mind:

- Keep the users informed about where they are. Either use a tool tip or status area to communicate the range of data they are scrolling into.
- Give feedback while waiting on data to load.
- Create the illusion of an entire loaded virtual space for when the data feels like a data set (e.g., Yahoo! Mail's mail messages).
- Extend the virtual space during scroll for search results (e.g., Microsoft Live Search).
- Keep a limited virtual space when the user is moving through search results in order to provide a nice back-and-forth experience (e.g., PicLens).

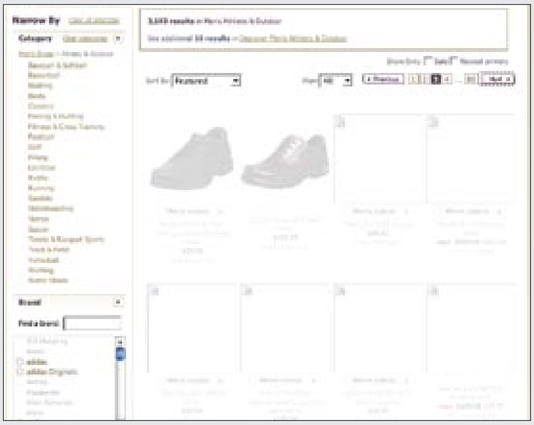
Inline Paging

What if instead of scrolling through content we just wanted to make pagination feel less like a page switch? By only switching the content in and leaving the rest of the page stable, we can create an **Inline Paging** experience. This is what Amazon's Endless.com site does with its search results (Figure 7-6).



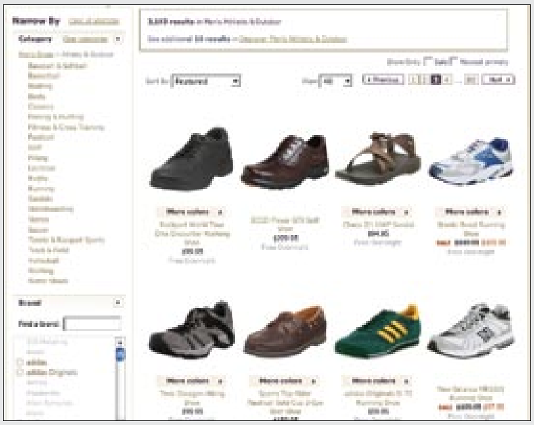
Paginated results

Searching for "Men's athletic shoes" displays a traditional-looking set of search results. The pagination controls are familiar (shown as an exploded callout).



Inline Paging

Clicking to "page 3" causes just the search results area to update with the third page of results.



Page remains stable

The rest of the page stays stable when the new "page" of results is brought into view.

Figure 7-6. *Endless uses Inline Paging to create a seamless experience moving through search results*

Considerations

There are some issues to consider with **Inline Paging**.

In-page update

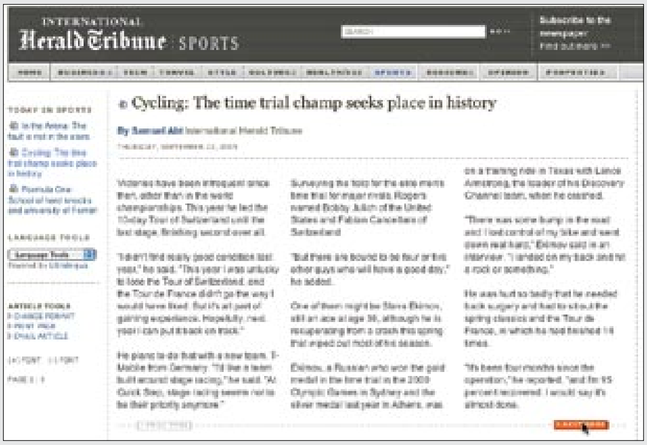
Endless.com provides the normal pagination controls. But instead of the whole page refreshing, only the results area is updated. Keeping the context stable creates a better flow experience. With **Inline Paging** it feels like the user never leaves the page even though new virtual pages of results are being brought into view.

Tip

Paging is sometimes the most convenient and natural way to break up information.

Natural chunking

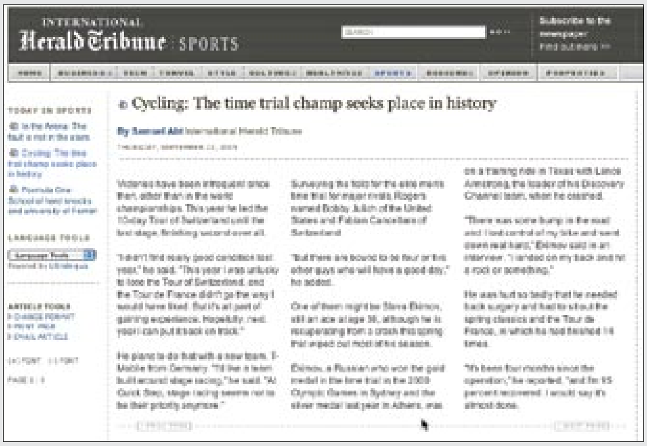
Inline Paging can also be useful when reading news content online. The *International Herald Tribune* applied this as a way to page through an article while keeping the surrounding context visible at all times (Figure 7-7).


INTERNATIONAL
Herald Tribune SPORTS
Cycling: The time trial champ seeks place in history
By Samuel Abt International Herald Tribune

Paging tool

The online version of the paper mimics the paper version.

But unlike the paper version, a "next page" button moves through the story.

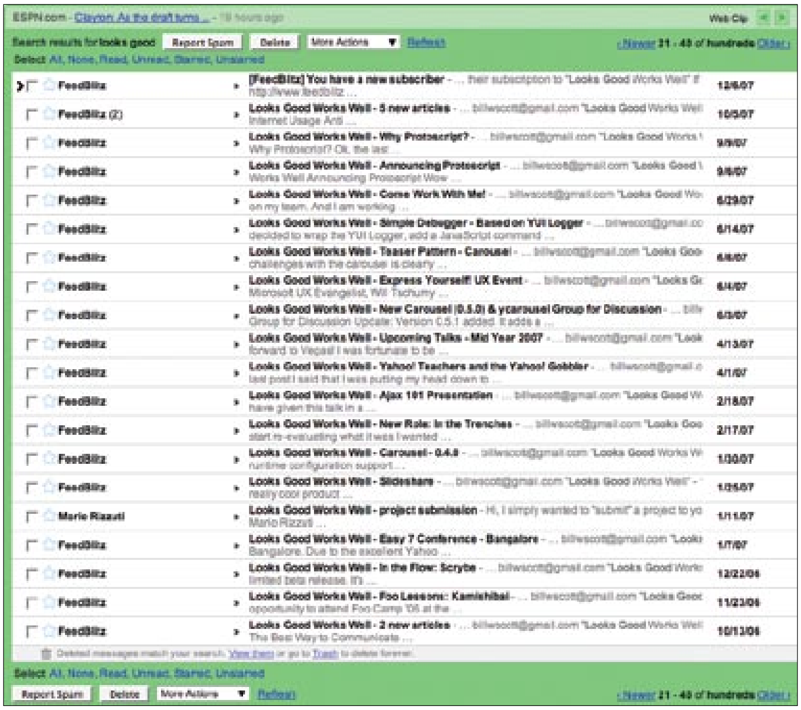

INTERNATIONAL
Herald Tribune SPORTS
Cycling: The time trial champ seeks place in history
By Samuel Abt International Herald Tribune

Inline Paging

Instead of refreshing the complete page, only the story columns are replaced with the next page.

Figure 7-7. *The International Herald Tribune uses Inline Paging to seamlessly move through a story without losing context*

Gmail also uses **Inline Paging** (Figure 7-8). A set number of messages is displayed on the page. Clicking the "Older" or "Newer" links moves the user to a new set of messages. However, instead of refreshing the complete page, just the message area updates. While there is still some interruption whenever the user switches pages, by making the transition "stay in the page" the interaction feels much lighter, and a virtual space is created through clever use of pagination.



ESPN.com - Clayton: As the draft turns ... - 19 hours ago   Web Clip

Search results for looks good   Report Spam   Delete   More Actions   Refresh   « Newer 21 - 40 of hundreds Older »

Select: All, None, Read, Unread, Starred, Unstarred

FeedBlitz	**[FeedBlitz] You have a new subscriber** - ... their subscription to "Looks Good Works Well" If http://www.feedblitz ...	12/6/07
FeedBlitz (2)	**Looks Good Works Well - 5 new articles** - ... billwscott@gmail.com "Looks Good Works Well Internet Usage Anti ...	10/3/07
FeedBlitz	**Looks Good Works Well - Why Protoscript?** - ... billwscott@gmail.com "Looks Good Works Why Protoscript? Ok, the last ...	9/9/07
FeedBlitz	**Looks Good Works Well - Announcing Protoscript** - ... billwscott@gmail.com "Looks Good Works Well Announcing Protoscript Wow ...	9/6/07
FeedBlitz	**Looks Good Works Well - Come Work With Me!** - ... billwscott@gmail.com "Looks Good Wo on my team. And I am working ...	6/29/07
FeedBlitz	**Looks Good Works Well - Simple Debugger - Based on YUI Logger** - ... billwscott@gmail.co decided to wrap the YUI Logger, add a JavaScript command ...	6/14/07
FeedBlitz	**Looks Good Works Well - Teaser Pattern - Carousel** - ... billwscott@gmail.com "Looks Goo challenges with the carousel is clearly ...	6/6/07
FeedBlitz	**Looks Good Works Well - Express Yourself! UX Event** - ... billwscott@gmail.com "Looks Go Microsoft UX Evangelist, Will Tschumy ...	6/4/07
FeedBlitz	**Looks Good Works Well - New Carousel (0.5.0) & ycarousel Group for Discussion** - ... bill Group for Discussion Update: Version 0.5.1 added. It adds a ...	6/3/07
FeedBlitz	**Looks Good Works Well - Upcoming Talks - Mid Year 2007** - ... billwscott@gmail.com "Look forward to Vegas! I was fortunate to be ...	4/13/07
FeedBlitz	**Looks Good Works Well - Yahoo! Teachers and the Yahoo! Gobbler** - ... billwscott@gmail.c last post I said that I was putting my head down to ...	4/1/07
FeedBlitz	**Looks Good Works Well - Ajax 101 Presentation** - ... billwscott@gmail.com "Looks Good W have given this talk in a ...	2/18/07
FeedBlitz	**Looks Good Works Well - New Role: In the Trenches** - ... billwscott@gmail.com "Looks Goo start re-evaluating what it was I wanted ...	2/17/07
FeedBlitz	**Looks Good Works Well - Carousel - 0.4.0** - ... billwscott@gmail.com "Looks Good Works W runtime configuration support ...	1/30/07
FeedBlitz	**Looks Good Works Well - Slideshare** - ... billwscott@gmail.com "Looks Good Works Well" - really cool product ...	1/25/07
Mario Rizzuti	**Looks Good Works Well - project submission** - Hi, I simply wanted to "submit" a project to yo Mario Rizzuti ...	1/11/07
FeedBlitz	**Looks Good Works Well - Easy 7 Conference - Bangalore** - ... billwscott@gmail.com "Looks Bangalore. Due to the excellent Yahoo ...	1/7/07
FeedBlitz	**Looks Good Works Well - In the Flow: Scrybe** - ... billwscott@gmail.com "Looks Good Works limited beta release. It's ...	12/22/06
FeedBlitz	**Looks Good Works Well - Foo Lessons: Kamishibai** - ... billwscott@gmail.com "Looks Good opportunity to attend Foo Camp '06 at the ...	11/23/06
FeedBlitz	**Looks Good Works Well - 2 new articles** - ... billwscott@gmail.com "Looks Good Works Well The Best Way to Communicate ...	10/13/06

Deleted messages match your search. View them or go to Trash to delete forever.

Select: All, None, Read, Unread, Starred, Unstarred

Report Spam   Delete   More Actions   Refresh   « Newer 21 - 40 of hundreds Older »

Figure 7-8. *Gmail paginates seamlessly by only updating the message area*

Sometimes **Inline Paging** can be as simple as clicking on a button to load more items into the page on demand. The newly loaded content can be added to the current page. This allows the content to be scrolled, but places the control outside the scrollbar.

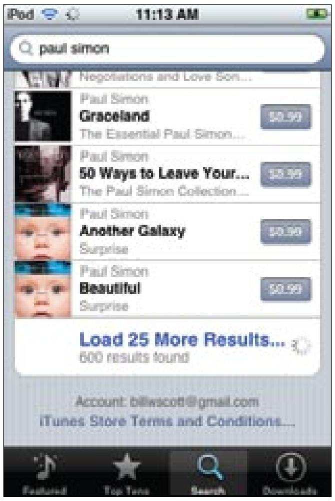
Back button

The biggest issue with **Inline Paging** is whether the back button works correctly. One criticism of Endless.com is that if the user pages through search results and then hits the back button, it jumps to the page just before the search. This unexpected result could be fixed by making the back button respect the virtual pages as well. This is the way Gmail handles the back button.[*] Clicking back moves you through the virtual pages.

Interactive content loading

The iPhone employs inline paging when displaying search results in the iTunes store (Figure 7-9).

[*] This was not always the case. Originally, Gmail did not handle the back button correctly. In the most recent versions, however, it has provided correct handling for the back button.



Negotiations and Love Son...

Paul Simon
Graceland
The Essential Paul Simon...
$0.99

Paul Simon
50 Ways to Leave Your...
The Paul Simon Collection...
$0.99

Paul Simon
Another Galaxy
Surprise
$0.99

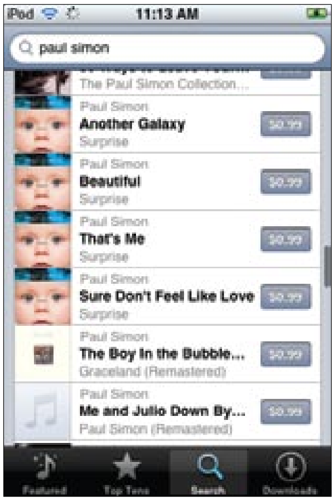
Paul Simon
Beautiful
Surprise
$0.99

Load 25 More Results...
600 results found

Account: billwscott@gmail.com
iTunes Store Terms and Conditions...

Figure 7-9. *The iPhone iTunes store displays 25 songs initially; the "Load 25 More Results…" button fetches 25 more songs*

Each tap of the "Load 25 More Results…" button loads 25 more songs. The number of songs loaded is cumulative. With the first tap there are now 50 songs; the third tap, 75 songs; and so on. Normally no scrollbar is shown. However, if the user places a finger on the list to move up or down, the scrollbar is displayed (Figure 7-10). Since the finger is the "scroller," the scrollbar becomes just an indicator of how many songs are loaded and where the user is in the list.



The Paul Simon Collection...

Paul Simon
Another Galaxy
Surprise
$0.99

Paul Simon
Beautiful
Surprise
$0.99

Paul Simon
That's Me
Surprise
$0.99

Paul Simon
Sure Don't Feel Like Love
Surprise
$0.99

Paul Simon
The Boy In the Bubble...
Graceland (Remastered)
$0.99

Paul Simon
Me and Julio Down By...
Paul Simon (Remastered)
$0.99

Figure 7-10. *The scrollbar displays when the user starts scrolling through the content; the scrollbar is just a feedback mechanism to indicate scrolling is happening, how much content is loaded, and where the user is in the scroll operation*

Best Practices for Inline Paging

Here are some best practices to keep in mind:

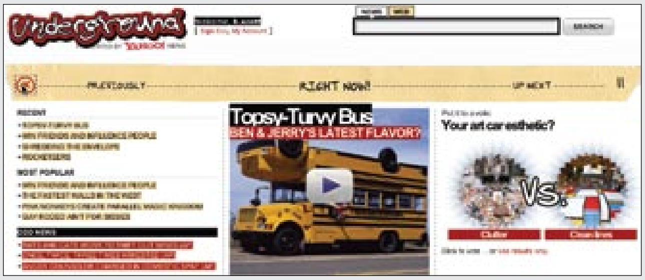
- Use **Inline Paging** for material that is naturally "chunked" but for which you still want to create a smooth viewing experience when moving between pages.
- Respect the back button. Make it work for paging (e.g., Gmail).
- Only update the "virtual page" and not the entire page when inline paging.
- Consider progressive loading as a way to page-in more content in a virtual space.

Scrolled Paging: Carousel

Besides **Virtual Scrolling** and **Virtual Paging**, there is another option. You can combine both scrolling and paging into **Scrolled Paging**. Paging is performed as normal. But instead the content is "scrolled" into view.

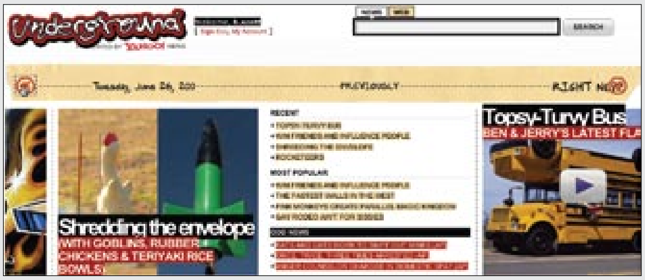
The **Carousel** pattern takes this approach. A **Carousel** provides a way to page-in more data by scrolling it into view. On one hand it is a variation on the **Virtual Scrolling** pattern. In other ways it is like **Virtual Paging** since most carousels have paging controls. The additional effect is to animate the scrolled content into view.

Yahoo! Underground uses a **Carousel** (Figure 7-11) to provide a way to page/scroll through articles.



Timeline

The top section provides a navigation control through various Underground articles. "Previously" and "Up Next" indicate where the user can go.



Animation

Animation reinforces the fact that the articles are from the past (the content moves in from the left to the right).

Figure 7-11. *Yahoo! Underground uses a Carousel to communicate a timeline of blog articles*

Considerations

There are some issues to consider when using **Scrolled Paging**.

Time-based

Carousels work well for time-based content. Flickr employs a **Carousel** to let users navigate back and forth through their photo collection (Figure 7-12).

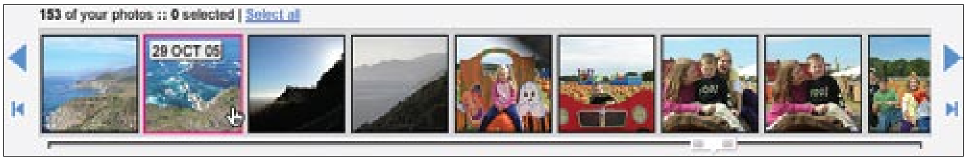


Figure 7-12. *Flickr uses a Carousel to allow the user to get photos in chronological order; this makes it possible to find photos without leaving the current page*

Tip

Carousels are best for featured or recent content. They are also good for small sets of time-based content.

Animation direction

Inexplicably, AMCtheatres.com animates its **Carousel** the opposite way. This leads to a confusing experience, and it's harder to know which control to click (Figure 7-13).

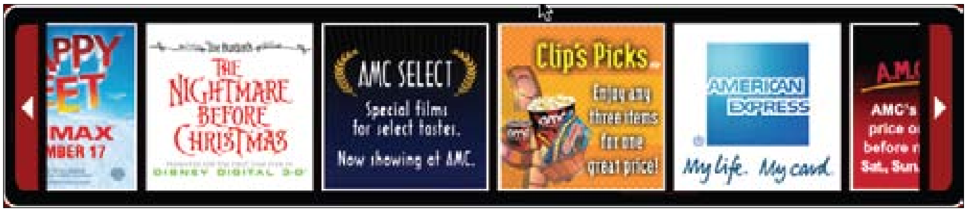

PPY
EET
MAX
MBER 17
THE NIGHTMARE BEFORE CHRISTMAS
DISNEY DIGITAL 3-D
AMC SELECT
Special films for select tastes.
Now showing at AMC.
Clip's Picks
Enjoy any three items for one great price!
AMERICAN EXPRESS
My life. My card.
AMC
AMC's price o before n Sat., Sun.

Figure 7-13. *AMC animates its carousel backward from the accepted standard: clicking the right arrow should scroll in content from the right (not the left)*

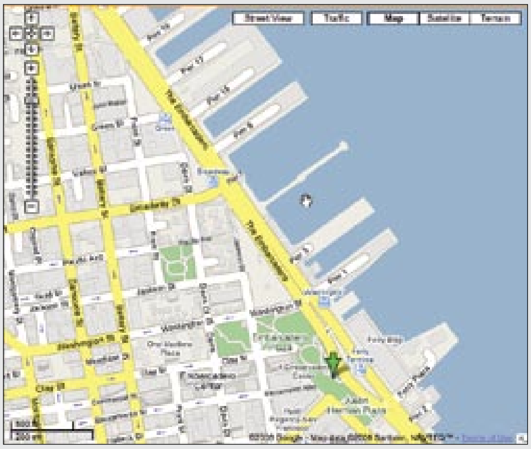
Carousel Best Practices

Here are some best practices to keep in mind:

- **Carousels** are best for visually distinct contents. Images, CD covers, and movie box shots are all natural items to place in a carousel.
- If the content is not in a discernible order, users will get frustrated navigating.
- If the content is highly relevant at the beginning and relevancy drops off, a **Carousel** is a good solution, since it spotlights the most relevant items.
- If there is a lot of content to display, carousels provide too small of a window and thus can frustrate users when they actually try to find content.
- If there is limited space available for a set of content, a **Carousel** allows the virtual content to be placed in a row on the page.
- Placing arrows at each end can become tedious if the **Carousel** is not circular or if the user needs to go back and forth through the content.
- Placing back and forth arrows next to each other is simpler to operate but somewhat less discoverable. It solves the back and forth issue but moves the buttons away from their naturally expected position (think scrollbars).
- Make the content big enough to view easily, providing enough whitespace to make each item distinct.
- Allow a portion of the next item to be partially revealed. This invites the user to scroll the content to see the partial content fully.
- Clicking the left arrow should scroll content in from the left (left to right), and clicking the right arrow should scroll content in from the right (right to left).

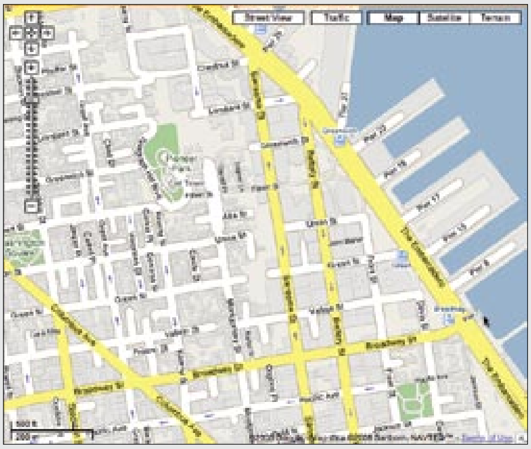
Virtual Panning

One way to create a virtual canvas is to allow users the freedom to roam in two-dimensional space. A great place for **Virtual Panning** is on a map. Google Maps allows you to pan in any direction by clicking the mouse down and dragging the map around (Figure 7-14).



Start drag

Clicking and holding down changes the cursor into a hand (signifying panning).



Dragging

Dragging the canvas moves the map in real time.

Figure 7-14. *Google Maps creates a virtual canvas; one tool that helps with that illusion is the ability to pan from area to area*

Considerations

There are some issues to consider while using **Virtual Panning**.

Natural Visual Construct

When we discussed dragging routes in Google Maps, we pointed out that drag and drop worked well since it fit with the natural visual representation of routes on the map (see Figure 2-26). In the same way, panning around in a map is a natural visual metaphor. Extending the visual space to a larger virtual space is a natural fit.

Gestures

Besides map applications, the idea of **Virtual Panning** has been extended to other devices thanks to gesture-based interfaces.* With the introduction of the iPhone, the user can simply "flick" through weather locations, photos, or an iTunes playlist.

Flicking is similar to panning yet has some differences. With **Virtual Panning** the canvas only moves while the mouse is dragging it around. With flicking, if the user starts the dragging operation and releases, the canvas will continue moving with some momentum. The canvas slows in such a way as to mimic real-world forces.

Best Practices for Virtual Panning

Here are some best practices to keep in mind:

- Use **Virtual Panning** as an alternative to scrolling through an infinite 2D canvas.
- Consider allowing panning to continue with some momentum (think "flicking").

Zoomable User Interface

A **Zoomable User Interface** (ZUI) is another way to create a virtual canvas. Unlike panning or flicking through a flat, two-dimensional space, a **ZUI** allows the user to also zoom in to elements on the page. This freedom of motion in both 2D and 3D supports the concept of an infinite interface.

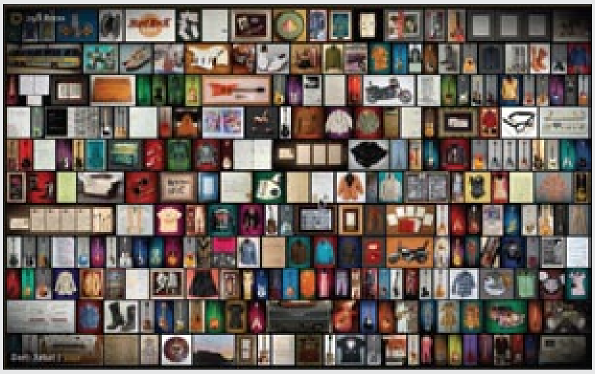
Practically speaking, **ZUI**s have rarely been available in everyday software applications, much less on the Web. But with more advanced features added to Flash and the advent of Silverlight, this type of interface is starting to emerge and may be commonplace in the not-too-distant future.

Considerations

Hard Rock Café has a large rock and roll memorabilia collection. Recently, it digitized photos of the collection and placed them online for virtual viewing.† The memorabilia application uses a **ZUI** interface to move around from artifact to artifact and zoom in to see details on each item (Figure 7-15).

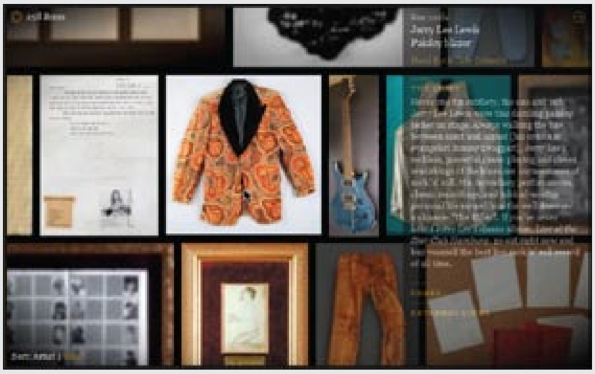
* For a discussion on gestural interfaces, see *Designing Gestural Interfaces* by Dan Saffer (O'Reilly).

† The Hard Rock Memorabilia site is located at *http://memorabilia.hardrock.com/*. It was developed in Silverlight using Deep Zoom (a.k.a. Seadragon) by Vertigo Software in Point Richmond, CA.



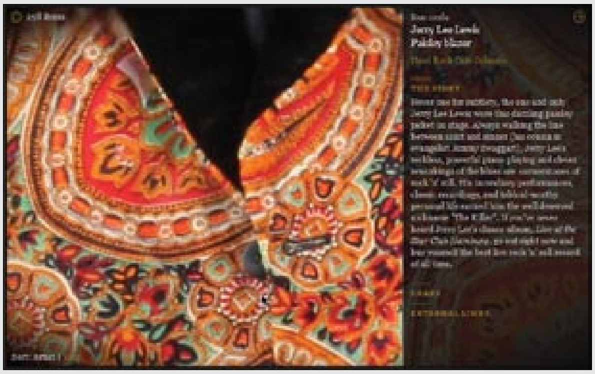
Zoomed-out

At the zoomed-out level the user can see thumbnails of the total collection.



Zooming in

By using the mouse thumb-wheel, the user can zoom in (it is like flying) on any object.

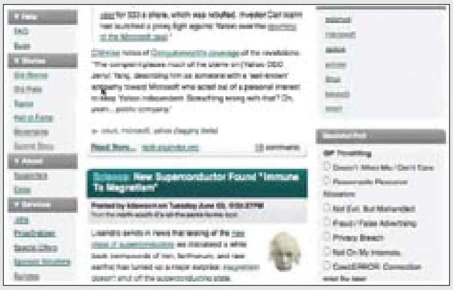


Detail stitched in

As the user gets closer and closer, more detail is stitched in. Detail is only limited by what can be dynamically mapped in as the user zooms.

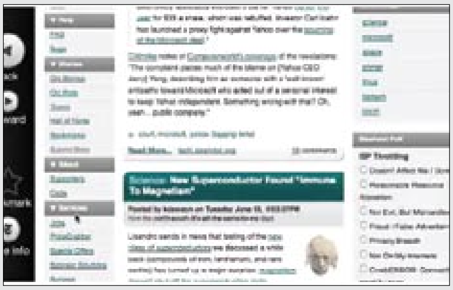
Figure 7-15. *Hard Rock Café uses a zoomable user interface (ZUI) to allow its memorabilia collection to be viewed online*

Aza Raskin, son of the late Jef Raskin (who pioneered many of the original ZUI concepts) is continuing to experiment with user interfaces that push the current norms. He demonstrated some potential **ZUI** interactions in a concept demo for Firefox on the mobile (Figure 7-16).*



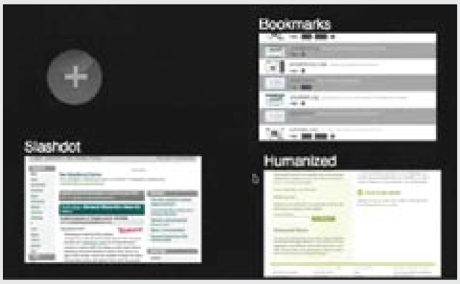
Zoomed-in to content

This browser page lives on a large canvas. This view is zoomed in to the page.



Slide over

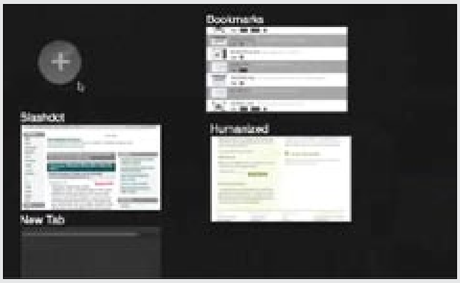
Using a panning technique, the page is pulled to the right, revealing a hidden toolbar on the left.



Zoomed-out

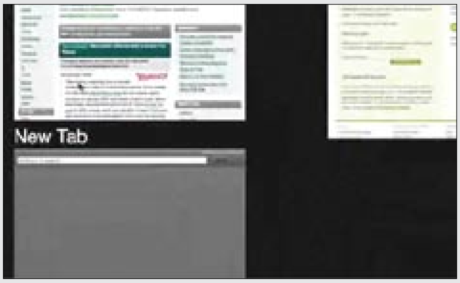
The canvas can contain many "Tabs", or windows.

* You can see Aza's concept video at *http://www.azarask.in/blog/post/firefox-mobile-concept-video/*.



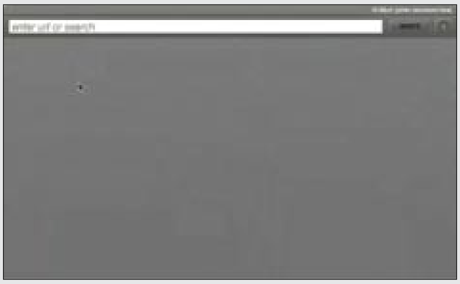
Add a Tab

Clicking the plus sign adds a new Tab. The interface zooms out slightly, revealing a new window being created in the bottom left of the canvas.



Zooming in to new Tab

The canvas automatically zooms into the newly created Tab.



New Tab

The new Tab now fills up the screen as the interface settles in to a zoomed-in mode.

Figure 7-16. *Firefox Mobile Concept Prototype showing panning gestures as well as a ZUI interface*

In this concept prototype, two key ideas are presented:

- Additional interface elements are just off the page to the top, left, bottom, and/or right. Pulling the content to one side or the other reveals these tools.
- Moving between the canvas (desktop) and windows is accomplished with a ZUI interface.

ZUIs provide the ultimate virtual canvas. By extending the concept of the page, the user never actually leaves the virtual page.

Best Practices for Virtual Spaces

Here are some best practices to keep in mind:

- Use a hybrid **Scrolled Paging** for content that is time-based.
- Use **Virtual Panning** for moving around in a canvas (e.g., Google Maps).
- Use **ZUI**s to work on large sets of information that have lots of detail.

Paging Versus Scrolling

Leading web designers and companies have taken different approaches to solving the same problems. Yahoo! Mail chose **Virtual Scrolling**. Gmail chose **Inline Paging**.

How do you choose between paging and scrolling? While there are no hard and fast rules, here are some things to consider when making the decision:

- When the data feels "more owned" by the user—in other words, the data is not transient but something users want to interact with in various ways. If they want to sort it, filter it, and so on, consider **Virtual Scrolling** (as in Yahoo! Mail).
- When the data is more transient (as in search results) and will get less and less relevant the further users go in the data, **Inline Paging** works well (as with the iPhone).
- For transient data, if you don't care about jumping around in the data to specific sections, consider using **Virtual Scrolling** (as in Live Image Search).
- If you are concerned about scalability and performance, paging is usually the best choice. Originally Microsoft's Live Web Search also provided a scrollbar. However, the scrollbar increased server-load considerably since users are more likely to scroll than page.
- If the content is really continuous, scrolling is more natural than paging.
- If you get your revenue by page impressions, scrolling may not be an option for your business model.
- If paging causes actions for the content to become cumbersome, move to a scrolling model. This is an issue in Gmail. The user can only operate on the current page. Changing items across page boundaries is unexpected. Changing items in a continuous scrolled list is intuitive.

Chapter 8
Process Flow

In the last three chapters we've been discussing the principle *Stay on the Page*. Sometimes tasks are unfamiliar or complicated and require leading the user step-by-step through a **Process Flow**. It has long been common practice on the Web to turn each step into a separate page. While this may be the simplest way break down the problem, it may not lead to the best solution. For some **Process Flows** it makes sense to keep the user on the same page throughout the process.

Google Blogger

The popular site Google Blogger generally makes it easy to create and publish blogs. One thing it does not make easy, though, is deleting comments that others may leave on your blog. This is especially difficult when you are the victim of hundreds of spam comments left by nefarious companies hoping to increase their search ranking.

Blogger forces you to delete these comments through a three-step process. Each step is an individual page, all punctuated with a page refresh (Figure 8-1).

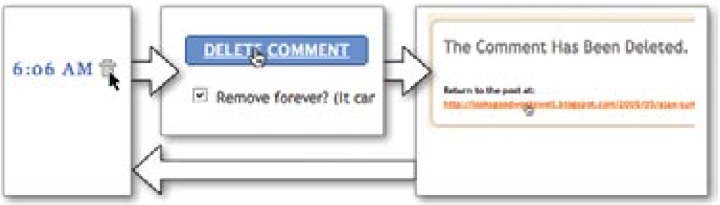


Figure 8-1. *Google Blogger forces you through a three-step process for each comment you delete, which is especially tiresome if you have dozens of spam comments to delete*

My (Bill's) blog site was recently spammed. It turns out that my 100 or so articles all had 4 or more spam comments. That means that I had to delete more than 400 spam comments. Given the way Google Blogger implemented comment deleting, I had to follow these steps for each comment on each blog article:

1. Scroll to find the offending comment.
2. Click the trash icon to delete the comment.
3. After page refreshes, click the "Remove Forever" checkbox.
4. Click the "Delete Comment" button.
5. After the page refreshes, click the link to return to my blog article.
6. Repeat steps 1–5 for each article with spam comments.

It took 1,600 clicks, 1,200 page refreshes, 400 scroll operations, and several hours to finally rid myself of all of the spam comments. If the delete action could have been completed in the same page as the comments, that would have eliminated hundreds of clicks and well over a thousand page refreshes, and scrolling would have been all but eliminated. I would not have wasted all the mental energy to reorient myself after each page transition. And I would have been a much happier man.

This is a common interaction flow on the Web. It turns out to be simpler to design and implement a process as a series of pages rather than a single interactive space.

The Magic Principle

Alan Cooper discusses a wonderful technique for getting away from a technology-driven approach and discovering the underlying mental model of the user. He calls it the "magic principle."[*] Ask the question, "What if when trying to complete a task the user could invoke some magic?" For example, let's look at the problem of taking and sharing photos. The process for this task breaks down like this:

- Take pictures with a digital camera.
- Sometime later, upload the photos to a photo site like Flickr. This involves:
  - — Finding the cable.
  - — Starting iTunes.
  - — Importing all photos.

[*] Discussed in his "Interaction Design Practicum" course, found at *http://www.cooper.com/services/training/ixd_practicum.html*.

— Using a second program, such as Flickr Uploadr, to upload the photos to Flickr.

— Copying the link for a Flickr set (which involves first locating the page for the uploaded set).

- Send the link in email to appropriate friends.

If some magic were invoked, here is how it might happen:

- The camera would be event-aware. It would know that is your daughter's eighth birthday.
- When finished taking pictures of the event, the camera would upload the pictures to Flickr.
- Flickr would notify family and friends that the pictures of the birthday party are available.

Thinking along these lines gets some of the artifacts out of the way. Of course the magic could be taken to the extreme: just eliminate the camera altogether! But by leaving some elements in the equation, the potentially unnecessary technology pieces can be exposed. How about the cable? What if the camera could talk magically to the computer?

This kind of thinking led to some recent products that allow users to upload photos automatically from their digital camera to their favorite photo site. The camera's memory card actually contains a Wi-Fi connection, giving it direct access to a photo-upload service.*

Turning back to our world, when we look at the multiple page **Process Flows** that litter the Web, we find that a little magic could dispense with the artifact of the "page" in many places. Amazon's shopping process illustrates this artifact. It provides a fairly typical multi-page experience when users add items to a cart (Figure 8-2).

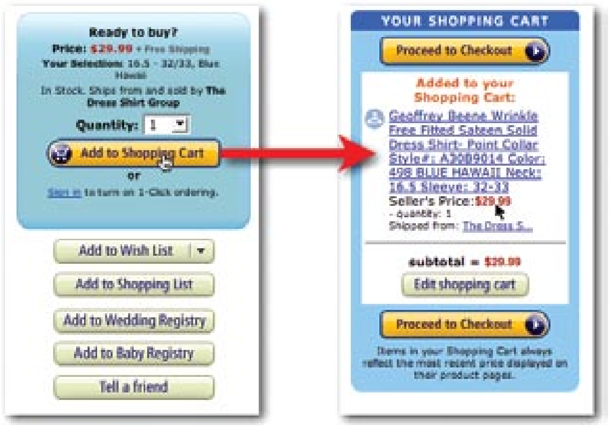

Ready to buy?
Price: $29.99 + Free Shipping
Your Selection: 16.5 - 32/33, Blue Hawaii
In Stock. Ships from and sold by The Dress Shirt Group
Quantity: 1
Add to Shopping Cart
or
Sign in to turn on 1-Click ordering.
Add to Wish List
Add to Shopping List
Add to Wedding Registry
Add to Baby Registry
Tell a friend
YOUR SHOPPING CART
Proceed to Checkout
Added to your Shopping Cart:
Geoffrey Beene Wrinkle Free Fitted Sateen Solid Dress Shirt- Point Collar Style#: A3009014 Color: 498 BLUE HAWAII Neck: 16.5 Sleeve: 32-33
Seller's Price: $29.99
- Quantity: 1
Shipped from: The Dress S...
subtotal = $29.99
Edit shopping cart
Proceed to Checkout
Items in your Shopping Cart always reflect the most recent price displayed on their product pages.

Figure 8-2. *Amazon's multi-page "Add to Shopping Cart" process*

* Eye-Fi is an example of this. See *http://www.eye.fi/overview/*.

First, the user finds a book. Then when she adds it to her shopping cart, she is taken to a separate page. Getting back happens either through the back button or finding the "Continue Shopping" button elsewhere on the page. Each book add is a heavy context switch to the second page.

But what if we could apply some magic? Could we get away with having fewer page transitions? Could we eliminate the multi-page transition?

Yes. It is possible to take flows like this and present them in a single interactive space. At other times, we can convert a multi-page experience into a single, long page. But sometimes the process is best left broken into individual pages. In those cases we can often encapsulate the multiple pages into a simple **Dialog Overlay**, which effectively maintains the context while preserving distinct page-to-page flow.

John Maeda, author of the book *Laws of Simplicity* (MIT Press), discusses the magic of hiding complexity:

> *Hiding complexity through ingenious mechanical doors or tiny display screens is an overt form of deception. If the deceit feels less like malevolence, more like magic, then hidden complexities become more of a treat than a nuisance.*

In this chapter we will apply a little magic, hopefully get rid of the nuisance of page transitions where possible, and in the end treat the user to a better experience. Specifically, we will look at these **Process Flow** patterns:

- **Interactive Single-Page Process**
- **Inline Assistant Process**
- **Configurator Process**
- **Overlay Process**
- **Static Single-Page Process**

Interactive Single-Page Process

Consumer products come in a variety of shapes, sizes, textures, colors, etc. Online shoppers will not only have to decide that they want shoes, but do they want blue suede shoes? And what size and width do they want them in? In the end the selection is constrained by the available inventory. As the user makes decisions, the set of choices gets more and more limited.

This type of product selection is typically handled with a multi-page workflow. On one page, the user selects a shirt and its color and size. After submitting the choice, a new page is displayed. Only when the user arrives at this second page does he find out that the "true navy" shirt is not available in the medium size.

The Gap accomplishes this kind of product selection in a single page (Figure 8-3) using **Interactive Single-Page Process**. The purple shirt is available in all sizes from XS to XXXL. Hovering over the dark blue shirt immediately discloses that this color is only available in XS and S sizes.

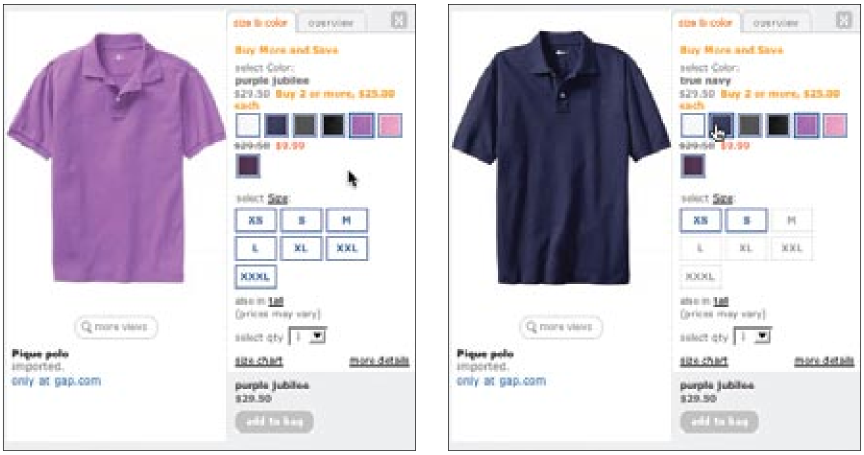


Figure 8-3. *The Gap uses Interactive Single-Page Process to reflect the sizes for each product color choice in real time*

Considerations

There are some issues to consider when using an **Interactive Single-Page Process**.

Responsiveness

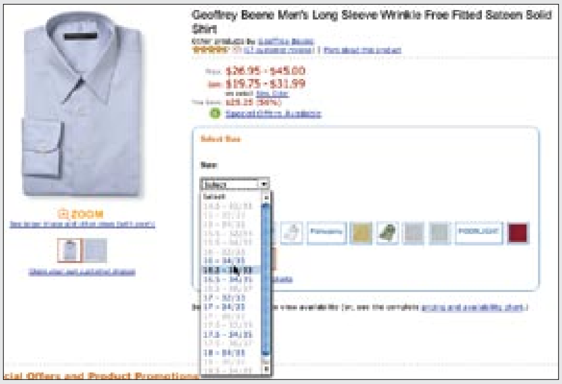
The user's taste preference comes first. Either the color or the size can be chosen. If the item is out of stock for any color/size combination, it is displayed as unavailable (by showing the color or size as disabled). By placing this process in a few simple interactions, the user can quickly find something available to buy. With any online shopping experience, the potential for the user to bail out is a real concern. In-place interactions like this reduce these *bailout moments*.

In *Designing Interfaces* (O'Reilly), Jenifer Tidwell uses the term **Responsive Disclosure** to describe this same pattern. According to Tidwell, **Responsive Disclosure** is good when:

> *The user should be walked through a complex UI task step-by-step, perhaps because he is computer-naive, or because the task is novel or rarely done (as in a Wizard). But you don't want to force the user to go page-by-page at each step—you'd rather keep the whole interface on one single page.*[*]

[*] See *http://www.oreilly.com/catalog/9780596008031* or page 123 in printed book.

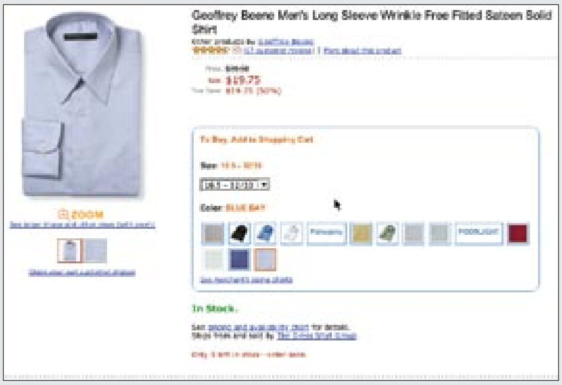
Amazon's interface for selecting a shirt also uses **Interactive Single-Page Process** with a slightly different interface (Figure 8-4).


Geoffrey Beene Men's Long Sleeve Wrinkle Free Fitted Sateen Solid Shirt
$26.95 - $45.00
$19.75 - $31.99
ZOOM
Select Size
Size:
Select
16.5 - 34/35
Special Offers and Product Promotions

Selecting sizes

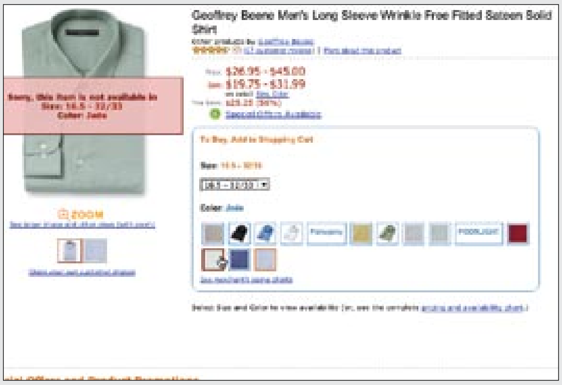
Sizes are shown in a drop-down menu. The sizes that are not available for the selected color are disabled.

If a disabled size is selected, the unavailable colors will be displayed in a dimmed state.


Geoffrey Beene Men's Long Sleeve Wrinkle Free Fitted Sateen Solid Shirt
$19.75
ZOOM
To Buy, Add to Shopping Cart
Size:
Color: BLUE BAY
In Stock.

Selecting color

The colors available for this size neck are clearly shown in the color selection palette.


Geoffrey Beene Men's Long Sleeve Wrinkle Free Fitted Sateen Solid Shirt
$26.95 - $45.00
$19.75 - $31.99
Sorry, this item is not available in
Size: 16.5 - 32/33
Color: Jade
ZOOM
To Buy, Add to Shopping Cart
Color: Jade

Unavailable

If a combination that is unavailable is selected, a clear warning is displayed and no links to purchase are made available.

Figure 8-4. *Amazon provides a similar experience to the Gap, utilizing Interactive Single-Page Process*

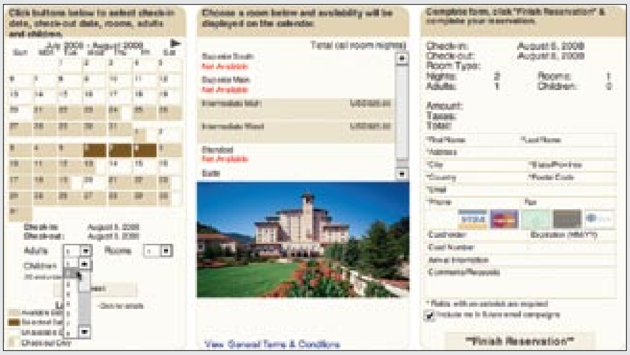
The idea is the same: make the experience for selecting a product painless by providing inventory disclosures as quickly as possible, and doing it all in a single-page interface.

Tip

Interactive, single-page process flows improve user engagement and increase conversion rates.

Keeping users engaged

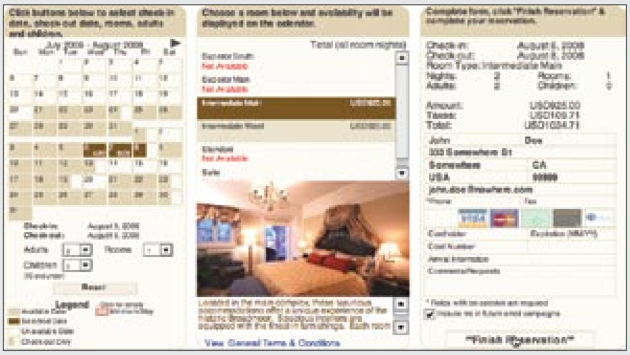
Broadmoor Hotel uses **Interactive Single-Page Process** for room reservations (Figure 8-5).



Date selection

In the first column, the calendar reveals available dates. Check-in and check-out dates can be chosen from a calendar.

Available room types are shown for the date and number of people.



Room and payment

The available rooms are shown in the second column.

Selecting a room fills out price information on the right.

Credit card contact information is filled out on the right to hold a reservation.

Figure 8-5. *Broadmoor Hotels provides a one-page interactive reservation system*

Each column represents what would normally be presented on a separate page. In the first column, a calendar discloses availability up front. This prevents scheduling errors. Selecting the room from the second column updates both the room picture and the pricing. The pricing is reflected back on the calendar days (Figure 8-6) as well as in the third column where credit card and contact information is entered.

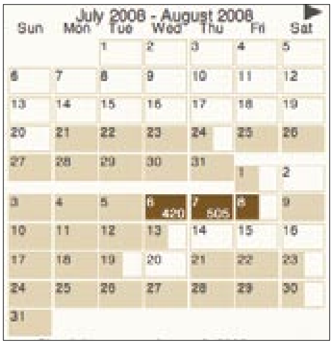


Figure 8-6. *The Broadmoor calendar shows availability and pricing information*

Benefits

Adobe calls out the Broadmoor one-page reservation interface in its Adobe Showcase.* It states the benefits of this method:

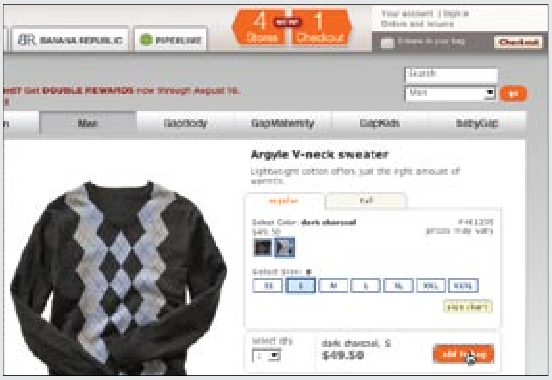
- Reduces entire reservation process to a single screen.
- Reduces the number of screens in the online reservation process from five to one. Other online reservation applications average 5 to 10 screens.
- Seventy-five percent of users choose OneScreen in favor of the HTML version.
- Allows users to vary purchase parameters at will and immediately view results.
- Reduces the time it takes to make a reservation from at least three minutes to less than one.

Additionally, Adobe notes that conversion rates (users who make it through the reservation process) are much higher with the **Interactive Single-Page Process**.

Inline Assistant Process

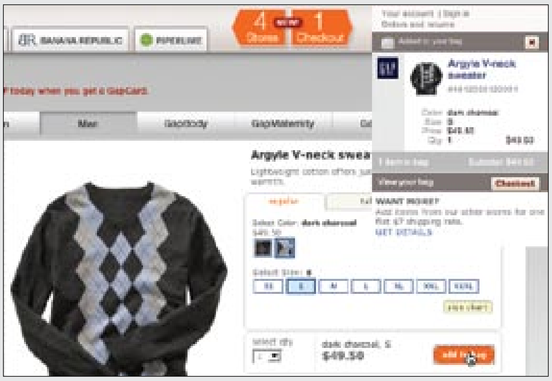
Another common place where multiple pages are used to complete a process is when adding items to a shopping cart. As mentioned earlier, Amazon provides the typical experience (Figure 8-2). So what magic can we apply to move this from a multi-page experience to a single-page experience? Instead of thinking about the cart as a process, we can think about it as a real-world object. Given this mindset, the cart can be realized in the interface as an object and be made available on the page. The Gap employed an **Inline Assistant Process** pattern for its shopping cart when it re-launched its site a few years back (Figure 8-7).

* Other benefits are highlighted in the case study. See *http://www.adobe.com/cfusion/showcase/index.cfm?event=casestudyprint&casestudyid=2486*.



"Add to Bag" button in context

"Add to Bag" is available in the context of selecting the sweater pattern and size.



Inline shopping cart

The shopping cart is not a separate page. Instead it is an interface element that is tied to each page.

It is always available as a drop-down shade. It is readily available to add items to or to open up and manage at any time.

Figure 8-7. *The Gap's single-page "Add to Bag" process*

Considerations

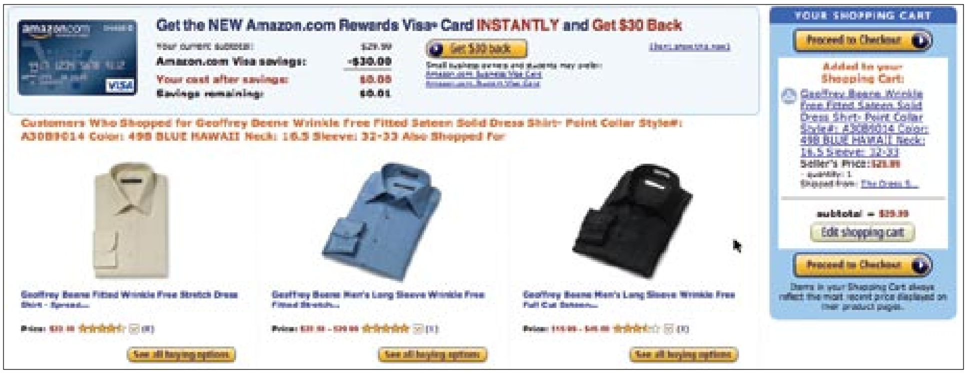
There are some things to consider when using the **Inline Assistant Process**.

Quick and easy

The Gap integrates the shopping cart into its entire site as a drop-down shade. In fact, the Gap, Old Navy, Banana Republic, and PiperLime all share the same **Inline Assistant Process**-style shopping cart. The Gap is betting that making it quick and easy to add items to the cart across four stores will equal more sales.

Additional step

Amazon, on the other hand, is betting on its recommendation engine. By going to a second page, Amazon can display other shirts like the one added—as well as advertise the Amazon.com Visa card (Figure 8-8).



Get the NEW Amazon.com Rewards Visa Card INSTANTLY and Get $30 Back

Your current subtotal: $29.99
Amazon.com Visa savings: -$30.00
Your cost after savings: $0.00
Savings remaining: $0.01

Customers Who Shopped for Geoffrey Beene Wrinkle Free Fitted Sateen Solid Dress Shirt- Point Collar Style#: A30B9014 Color: 498 BLUE HAWAII Neck: 16.5 Sleeve: 32-33 Also Shopped For

Geoffrey Beene Fitted Wrinkle Free Stretch Dress Shirt - Spread...

Geoffrey Beene Men's Long Sleeve Wrinkle Free Fitted Stretch...

Geoffrey Beene Men's Long Sleeve Wrinkle Free Full Cut Sateen...

YOUR SHOPPING CART

Added to your Shopping Cart:

Geoffrey Beene Wrinkle Free Fitted Sateen Solid Dress Shirt- Point Collar Style#: A30B9014 Color: 498 BLUE HAWAII Neck: 16.5 Sleeve: 32-33

subtotal = $29.99

Items in your Shopping Cart always reflect the most recent price displayed on their product pages.

Figure 8-8. *Amazon shows recommendations when confirming an add to its shopping cart*

Which is the better experience? The Gap seems to be the clear winner in pure user experience. But which brings in more money? It's a question we cannot answer, but the right one for any site to ask.

Tip

Use a quick and easy approach to get users to finish a task. Use additional steps when there are opportunities to engage the user further (as with recommendations).

Blending quick and easy with the additional step

Is there a way to combine both the single-page experience and the recommendation experience? This is what Netflix does when a user adds movies to his shipping queue (Figure 8-9).

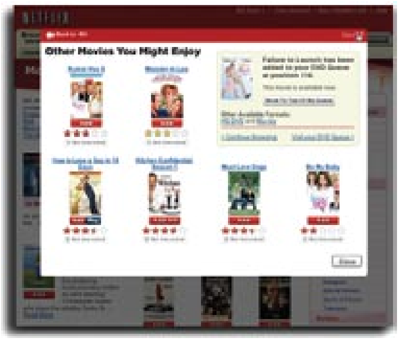


Figure 8-9. *Netflix displays its recommendations in an overlay*

Each movie on the site has an "Add" button. Clicking "Add" immediately adds the movie to the user's queue. As a confirmation and an opportunity for recommendations, a **Dialog Overlay** is displayed on top of the movie page.

Just like Amazon, Netflix has a sophisticated recommendation engine. The bet is that since the user has expressed interest in an item (shirt or movie), the site can find other items similar to it to suggest. Amazon does this in a separate page. Netflix does it in an overlay that is easily dismissed by clicking anywhere outside the overlay (or by clicking the close button at the top or bottom).

In a previous version of Netflix (or if JavaScript is disabled), this becomes a multiple-page experience (Figure 8-10).

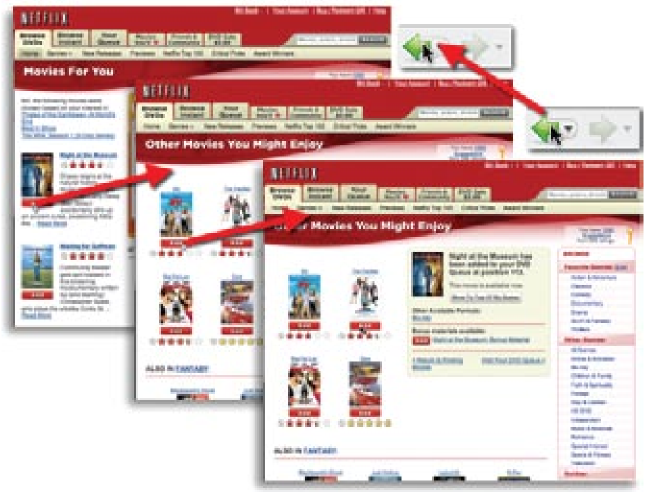


Figure 8-10. *Without JavaScript enabled, you can see the original multi-page flow for the "Add to Queue" process on the Netflix site*

Each movie add leads to a separate recommendation page. Clicking on "Add" for a movie on a recommendation page takes the user to a secondary recommendation page. This process can continue, on and on. Eventually, the user has to hit the back button a number of times to get back to the original context. In the **Overlay** example (Figure 8-9), the multi-page flow is encapsulated in the overlay and can be easily dismissed at any point, quickly returning the user to the originating page.

Best Practices for
Interactive Single-Page Process and Inline Assistant Process

Here are some best practices to keep in mind:

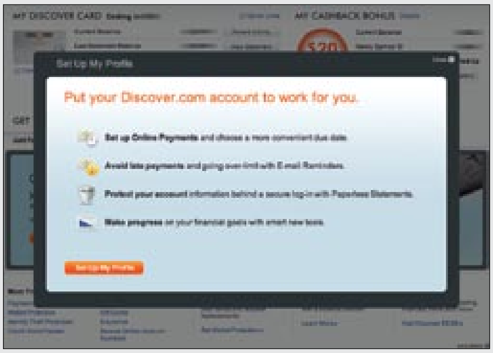
- Use these patterns to simplify user flow.
- Use these patterns to improve conversion rates.
- Show previews in context.
- Show invalid choices in real time. This informs users of the impact of their decisions.
- Show shopping carts inline where possible.
- Treat shopping carts as first-class interface elements and not just as an extra process.
- Show an additional step if recommendations are relevant and important.

Dialog Overlay Process

As mentioned before, any page switch is an interruption to the user's mental flow. In addition, any context switch is a chance for a user to leave the site. We seek an experience that has as little mental friction as possible. But sometimes the step-by-step flow is necessary.

The Netflix approach just described (Figure 8-9) uses a **Dialog Overlay Process** to encapsulate a multi-step flow inside a **Dialog Overlay**. We looked at **Overlays** in detail in Chapter 5. **Overlays** allow us to keep the context of the page yet present a virtual space to conduct a conversation with the user.

Discover.com recently expanded its account section with a more detailed profile. The profile captures things like your payment date, mobile fraud alerts, paperless statements, and general contact information (Figure 8-11). The overlay pops up when you first enter your account.


Set Up My Profile

Put your Discover.com account to work for you.

Set up Online Payments and choose a more convenient due date.

Avoid late payments and going over-limit with E-mail Reminders.

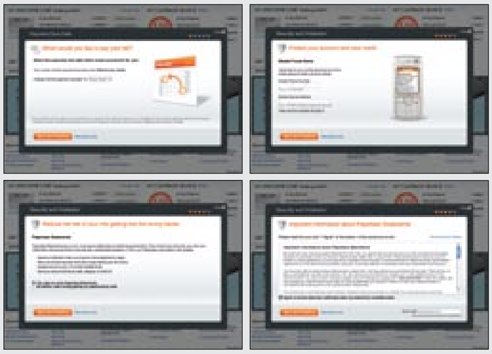
Protect your account information behind a secure log-in with Paperless Statements.

Make progress on your financial goals with smart new tools.

Invitation to set up profile

When entering an account, a **Dialog Overlay** is presented with an invitation to set up a profile.

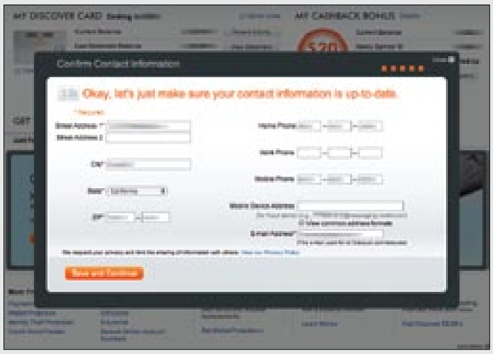
A **Lightbox Effect** is used to focus the user on the task of profile setup. The pleasing visuals add a level to the engagement factor.



Multiple steps

Each step is presented is presented as a separate "page" within the overlay.

Each "page" has a simple, clear call to action.


MY DISCOVER CARD
MY CASHBACK BONUS
Confirm Contact Information
Okay, let's just make sure your contact information is up-to-date.
Home Phone
Work Phone
Mobile Phone
Save and Continue

Final step

Filling out the contact information is left until the end. If this step occurs too early in the process, it might make the user think all the steps will be this involved.

Figure 8-11. *Discover encapsulates its "Create Your Profile" flow in a Dialog Overlay*

Considerations

There are some issues to consider when using the **Dialog Overlay Process**.

Making it feel lightweight

The **Lightbox Effect** is nice because it focuses the user on the task. But what is smart about the flow is the first page is colorful and rendered in a simple, pleasing manner. The process looks like it might be simple and feels engaging.

The in-between steps are usually a single action. There is a whole page dedicated just to selecting the due date for payments (Figure 8-12).

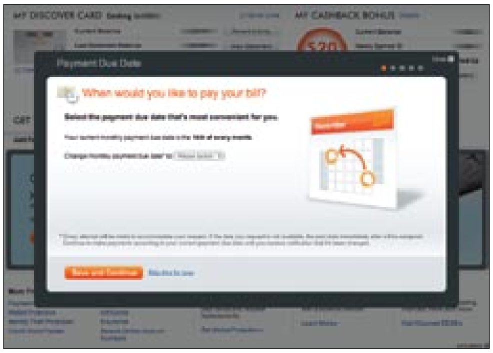

MY DISCOVER CARD
MY CASHBACK BONUS
Payment Due Date
When would you like to pay your bill?
Select the payment due date that's most convenient for you.

Figure 8-12. *Discover dedicates a complete "page" to just selecting when you would like to pay your bill*

Making the in-between steps clear and visually appealing with a single call to action makes the process feel lightweight. The last step is the biggest. By this time the user has committed to the process to some degree. Most of the user information is already filled in from the account, so the step does not feel too involved. But imagine if this step were first. It could cause users to more frequently bail out from the process, thinking that each step would be as involved as the first one.

Clear status

The other interesting touch in this example is providing an indication of the number of steps (Figure 8-13). There are any number of ways to indicate this information; the important thing is to give some indication of what the users are dealing with when they start. Usually three steps are ideal. In this case, there are five steps. But as we mentioned, the early steps are single actions.

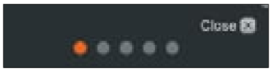
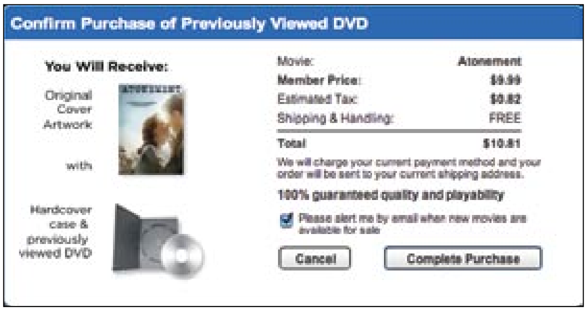


Figure 8-13. *Discover uses the orange dot to indicate how far the users are in the process and gray dots to indicate what they have left*

Netflix probably has one of the simplest overlay flows. When a user purchases a previously viewed DVD, a one-page overlay appears. Since the only way for a user to get to this purchase dialog is by being a member, Netflix has all the information it needs (shipping and billing information, for example) to complete the purchase in a single step (Figure 8-14).



Confirm Purchase of Previously Viewed DVD

You Will Receive:

Original Cover Artwork with Hardcover case & previously viewed DVD

Movie:	Atonement
Member Price:	**$9.99**
Estimated Tax:	$0.82
Shipping & Handling:	FREE
Total	**$10.81**

We will charge your current payment method and your order will be sent to your current shipping address.

100% guaranteed quality and playability

✓ Please alert me by email when new movies are available for sale

Figure 8-14. *Netflix has a single-step purchase flow for previously viewed DVDs*

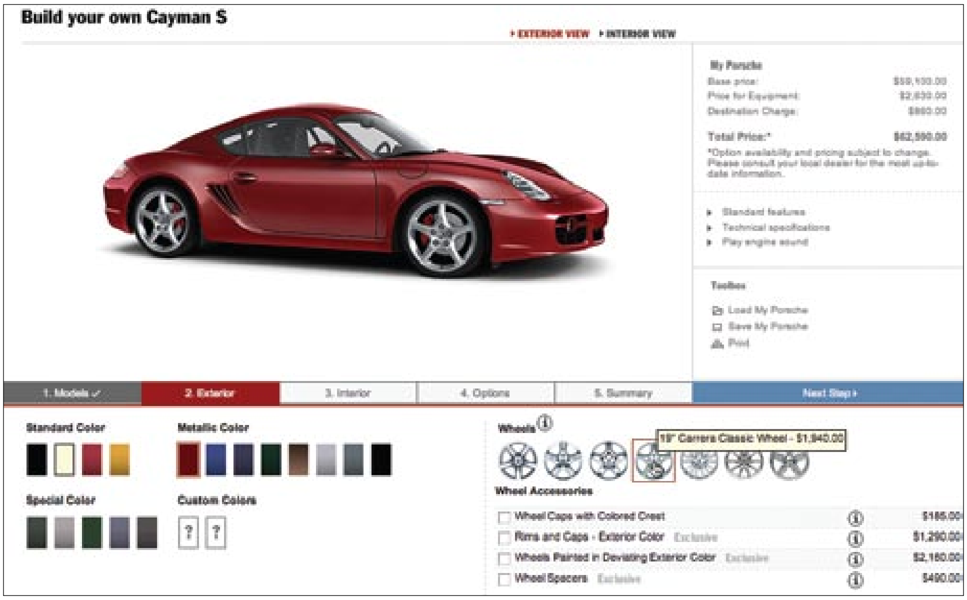
Best Practices for Dialog Overlay Process

Here are some best practices to keep in mind:

- Use **Dialog Overlay Process** to make multiple steps appear in context of the page.
- Use **Dialog Overlay Process** to make the process feel lighter than moving to different pages.
- Use a **Lightbox Effect** to focus users on the workflow.
- Keep the users informed throughout of their location in the process.

Configurator Process

Sometimes a **Process Flow** is meant to invoke delight. In these cases, it is the engagement factor that becomes most important. This is true with various **Configurator Process** interfaces on the Web. We can see this especially at play with car configurators. Porsche provides a configurator that allows users to build their own Porsche (Figure 8-15).



My Porsche

Base price:	$59,100.00
Price for Equipment:	$2,630.00
Destination Charge:	$860.00
Total Price:*	$62,590.00

*Option availability and pricing subject to change. Please consult your local dealer for the most up-to-date information.

Wheel Caps with Colored Crest	$185.00
Rims and Caps - Exterior Color Exclusive	$1,290.00
Wheels Painted in Deviating Exterior Color Exclusive	$2,160.00
Wheel Spacers Exclusive	$490.00

Figure 8-15. *Porsche's car configurator provides an engaging way to customize a car*

Being able to tweak the colors, wheels, interior, and options for a car and see the results in real time is an engaging experience. In Chapter 14, we discuss **Live Previews**. **Live Previews** allow the user to see the effect of his changes on a simulated version of the real thing.

Considerations

There are some issues to consider when using a **Configurator Process**.

Immediate feedback

In the case of the Porsche site, when the user clicks on various body colors, the car is re-configured in the selected color and rendered in photorealistic shading. Most configurators allow a 360-degree view of the car as it is configured.

Tip

Use immediate feedback (**Live Previews**) to simulate real-world choices.

Imagine if this was turned into a multi-page experience. Each panel would ask for the color, the wheel style, the interior material, dashboard configuration, and various other options. At the end, you would see your car. Not compelling? By making the experience engaging, users are more apt to experiment and fall in love with the car they have "designed."

It should be noted that the Porsche color configuration experience is actually part of a multi-page process flow. This highlights the fact that the decision between a single-page experience and a multi-page experience are not mutually exclusive. Single-page work-flows can live within a multi-page workflow.

Out of view status

Apple has a **Configurator Process** for purchasing a Macintosh computer (Figure 8-16).

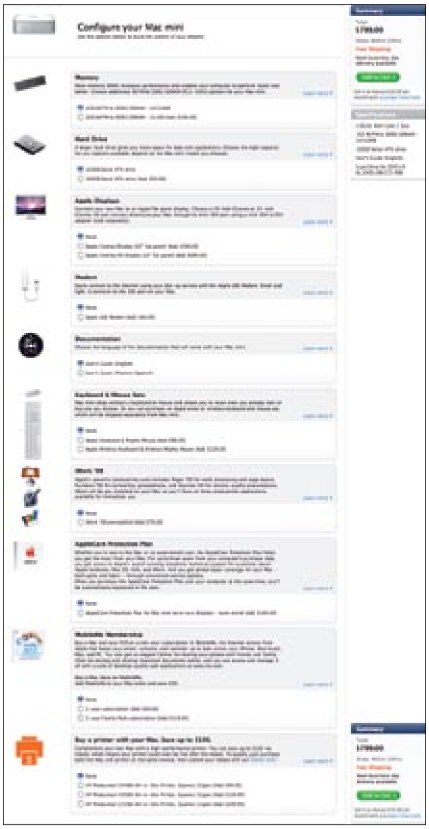


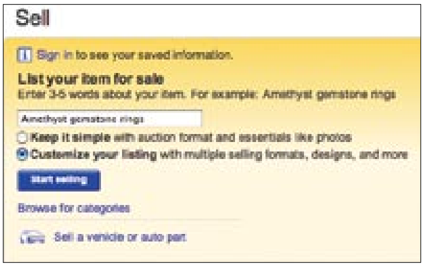
Figure 8-16. *Apple provides a one-page configurator for computer purchase*

There is a dynamic nature to this **Configurator Process**. As the user selects items to add to the computer, the price is dynamically updated using a **Spotlight** technique (discussed later in Chapter 11). In the upper-right corner of the page, the system configuration and price are highlighted to indicate updated information. The downside is that if this user is scrolled too far down, she can't see the update **Spotlight** since the summary panel is not visible.

Static Single-Page Process

The Apple example illustrates another way to get rid of multiple pages in a **Process Flow**. Just put the complete flow on one page in a **Static Single-Page Process**. The user sees all the tasks needed to complete the full process. This can be both good and bad. Seeing just one step to complete the process can encourage users to finish the task. But if the single step seems too long or too confusing, the user will most likely bail out of the process early. In other words, if placing all the tasks on a single page is enough to cause the user to bail out, it is not a good idea. In the case of the Apple store, each item is optionally set, and it's just a single click to include or exclude an item from the purchase.

eBay provides two ways to sell an item. An introductory panel (Figure 8-17) gathers the description of the item for sale. The "Customize your listings…" option takes the user through a traditional multi-page process (Figure 8-18).



Sell

Sign in to see your saved information.

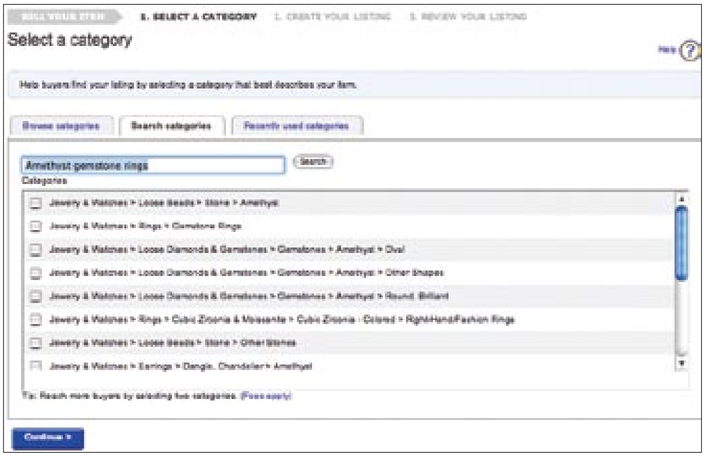
List your item for sale

Enter 3-5 words about your item. For example: Amethyst gemstone rings

Amethyst gemstone rings

Keep it simple with auction format and essentials like photos [✓ Customize your listing with multiple selling formats, designs, and more]

Figure 8-17. *eBay's simplified one-page flow*



Select a category

Help buyers find your listing by selecting a category that best describes your item.

Amethyst gemstone rings

Categories

- Jewelry & Watches > Loose Beads > Stone > Amethyst
- Jewelry & Watches > Rings > Gemstone Rings
- Jewelry & Watches > Loose Diamonds & Gemstones > Gemstones > Amethyst > Oval
- Jewelry & Watches > Loose Diamonds & Gemstones > Gemstones > Amethyst > Other Shapes
- Jewelry & Watches > Loose Diamonds & Gemstones > Gemstones > Amethyst > Round, Brilliant
- Jewelry & Watches > Rings > Cubic Zirconia & Moissanite > Cubic Zirconia - Colored > RightHand/Fashion Rings
- Jewelry & Watches > Loose Beads > Stone > Other Stones
- Jewelry & Watches > Earrings > Dangle, Chandelier > Amethyst

Tip: Reach more buyers by selecting two categories. (Fees apply)

Figure 8-18. *The customized flow is a traditional multi-page process*

Since the customized flow contains many options for listing an item, it requires a good deal of handholding. A multi-page process fits well with guiding the user through a complex set of tasks.

The other flow provided by eBay is a simplified **Static Single-Page Process** (Figure 8-19).

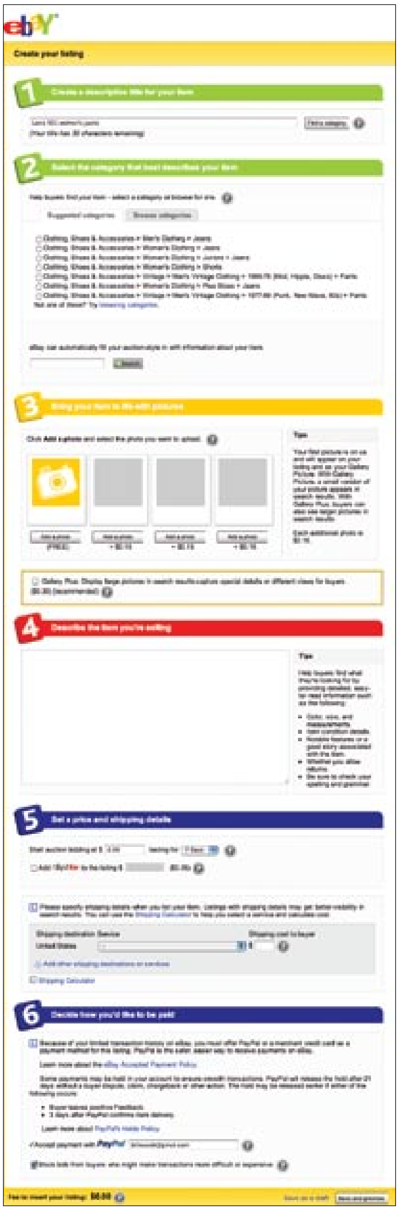


Figure 8-19. *eBay displays a simplified "Sell Your Item" as a Static Single-Page Process*

Considerations

There are some issues to keep in mind when using a **Static Single-Page Process**.

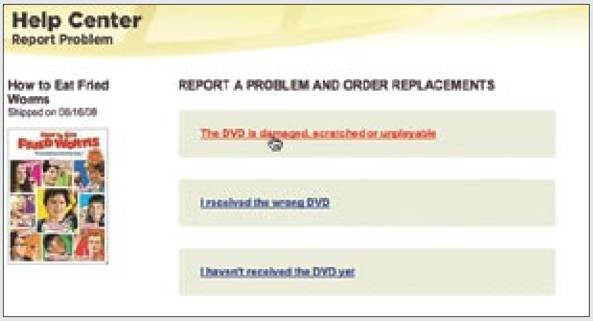
Making it feel lightweight

In this **Static Single-Page Process**, many options are defaulted and a simplified form is presented to the user. Multiple pages are compressed into a single page. Of course a long page like this can also be daunting. But eBay did a good job of getting the essentials into a single page. Each step is clearly called out with a clear border and a large step number. In addition, color is used to further simplify the steps. The two steps in green are about the item the user is selling. The last two steps in blue are about money. The color coding gives the impression there are only four steps total: item, photo, description, and pricing.

Multiple pages are not necessarily evil

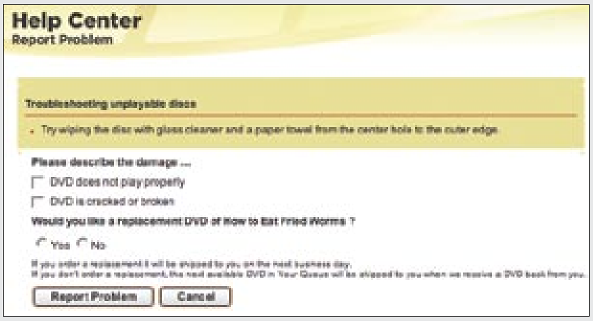
The eBay example (Figures 8-16 through 8-18) illustrates that there is more than one way to deal with a step-by-step **Process Flow**. For a very complex flow, a **Static Single-Page Process** may work well.

Sometimes it is good to break what could be a **Static Single-Page Process** into a multi-page process. Multiple pages can provide the natural chunking needed. They say "You are done with that step, now move onto the next." Netflix has a problem-reporting interface that does just that. When reporting a scratched disc, clicking on the "DVD is damaged…" link takes the user to a secondary page (Figure 8-20).


Help Center
Report Problem
How to Eat Fried Worms
Shipped on 08/16/08
REPORT A PROBLEM AND ORDER REPLACEMENTS
The DVD is damaged, scratched or unplayable
I received the wrong DVD
I haven't received the DVD yet

First page

The first page focuses the users on just deciding what the issue is.



Help Center
Report Problem

Troubleshooting unplayable discs

- Try wiping the disc with glass cleaner and a paper towel from the center hole to the outer edge.

Please describe the damage ...

- [ ] DVD does not play properly
- [ ] DVD is cracked or broken

Would you like a replacement DVD of How to Eat Fried Worms ?

○ Yes ○ No

If you order a replacement it will be shipped to you on the next business day.
If you don't order a replacement, the next available DVD in Your Queue will be shipped to you when we receive a DVD back from you.

Final page

The second page allows them to describe the details of that problem.

Figure 8-20. *Netflix provides a multi-page approach to reporting disc problems*

Why is it good to have a multi-page process in this case? Confirmation is a critical step in the process. The user needs a place to focus just on reporting a problem. Suggestions can be provided here without the context of all the other possible problems.

Best Practices for Static Single-Page Process

Here are some best practices to keep in mind:

- Use a multi-page process when the process is complex.
- Use a multi-page process when you want to hide the previous context (and the next) and bring the focus to a single task in the steps.
- Use a **Static Single-Page Process** when you only have a few steps and want to avoid taking the chance that a user will quit while moving from page to page.
- Use visual treatments to make the number of steps seem fewer.
- Provide clues as to where the user is and how much is left in a multi-step operation.
- Gather as many defaults as possible to simplify a flow.
- Use engagement, color visuals, interactivity, and simple visual styles to make the steps seem fewer.
- Put lightweight tasks up front in a multi-step operation.