15 User interface design

Objectives

The objective of this chapter is to introduce some aspects of user interface design that are important for software engineers. When you have read this chapter, you will:

- understand general design principles that should be followed by engineers responsible for user interface design;
- be aware of five different styles of interaction with a software system;
- have been introduced to different styles of information presentation and know when graphical presentation of information is appropriate;
- understand some fundamentals of the design of the user support that is embedded in software;
- understand usability attributes and simple approaches to system evaluation.

Contents

Computer system design encompasses a spectrum of activities from hardware design to user interface design. While specialists are often employed for hardware design, very few organisations employ specialist interface designers. Therefore, software engineers must often take responsibility for user interface design as well as for the design of the software to implement that interface. Human factors specialists may assist with this process in large organisations; in smaller companies such specialists are rarely used.

Good user interface design is critical to the success of a system. An interface that is difficult to use will, at best, result in a high level of user errors. At worst, users will simply refuse to use the software system irrespective of its functionality. If information is presented in a confusing or misleading way, users may misunderstand the meaning of information. They may initiate a sequence of actions that corrupt data or even cause catastrophic system failure.

When the first edition of this book was published in 1982, the standard interaction device was a 'dumb' alphanumeric terminal with green or blue characters displayed on a black background. User interfaces had to be textual or form-based. Almost all computer users now have a personal computer. These provide a graphical user interface (GUI) that supports a high-resolution colour display and interaction using a mouse as well as a keyboard.

Although text-based interfaces are still widely used, especially in legacy systems, computer users now expect application systems to have some form of graphical user interface. Figure 15.1 shows the principal characteristics of this type of interface.

The advantages of GUIs are:

1. They are relatively easy to learn and use. Users with no computing experience can learn to use the interface after a brief training session.
2. The user has multiple screens (windows) for system interaction. Switching from one task to another is possible without losing sight of information generated during the first task.

Figure 15.1
The characteristics of graphical user interfaces

Characteristic	Description
Windows	Multiple windows allow different information to be displayed simultaneously on the user's screen.
Icons	Icons represent different types of information. On some systems, icons represent files; on others, icons represent processes.
Menus	Commands are selected from a menu rather than typed in a command language.
Pointing	A pointing device such as a mouse is used for selecting choices from a menu or indicating items of interest in a window.
Graphics	Graphical elements can be mixed with text on the same display.

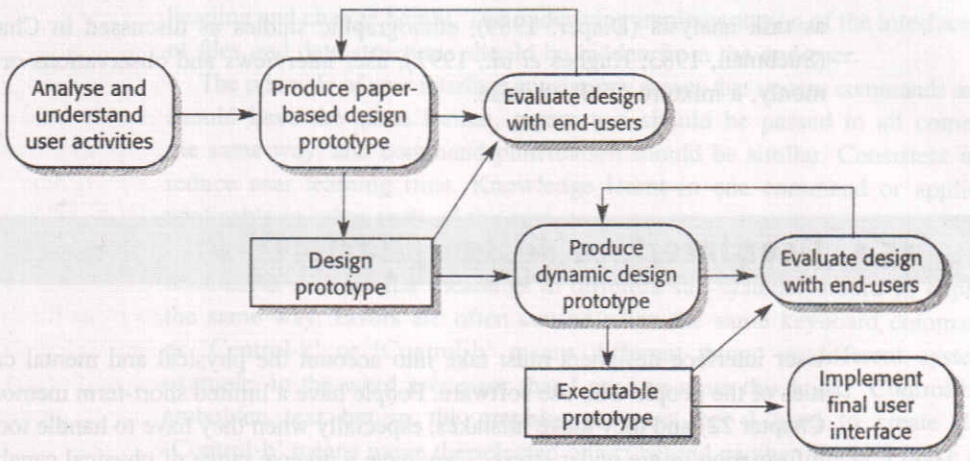


Figure 15.2
The user interface design process

3. Fast, full-screen interaction is possible with immediate access to anywhere on the screen.

The objective of this chapter is to sensitise software engineers to some of the key issues underlying user interface design. Software designers and programmers are often competent users of the technology, such as Java's Swing classes (Eckstein *et al.*, 1998) or HTML (Musciano and Kennedy, 1998), that is fundamental to user interface implementation. All too often, however, they do not use this technology in an appropriate way and create user interfaces that are inelegant, inappropriate and hard to use.

I focus, therefore, on giving some advice on the design of end-user facilities rather than the design of the software that implements these facilities. Because of space limitations, I only consider graphical interfaces. I do not cover interfaces that require special (perhaps very simple) displays such as mobile phones, video recorders, televisions, copiers and fax machines. Naturally, I can only introduce the topic here and I recommend texts such as those by Shneiderman (1998), Dix *et al.* (1998) and Preece *et al.* (1994) for more information on this topic.

Figure 15.2 illustrates the iterative process of user interface design. As I discuss in Chapter 8, exploratory development is the most effective approach to interface design. This prototyping process may start with simple paper-based interface mock-ups before going on to develop screen-based designs that simulate user interaction. A user-centred approach (Norman and Draper, 1986) should be used, with end-users of the system playing an active part in the design process. In some cases, the user's role is an evaluation one; in others, they participate as full members of the design team (Kyng, 1988; Greenbaum and Kyng, 1991).

A critical UI design activity is the analyses of the user activities that are to be supported by the computer system. Without an understanding of what the user wants to do with the computer system, there is no realistic prospect of designing an effective user interface. To develop this understanding, you may use techniques such

as task analysis (Diaper, 1989), ethnographic studies as discussed in Chapter 6 (Suchman, 1983; Hughes *et al.*, 1997), user interviews and observations or, commonly, a mixture of all of these.

15.1 User interface design principles

User interface designers must take into account the physical and mental capabilities of the people who use software. People have a limited short-term memory (see Chapter 22) and they make mistakes, especially when they have to handle too much information or are under stress. They have a diverse range of physical capabilities. You have to take all of these into account when designing user interfaces.

Human capabilities are the basis for the design principles shown in Figure 15.3. These are general principles which are applicable to all user interface designs and should normally be instantiated as more detailed design guidelines for specific organisations or types of system. A longer list of more specific user interface design guidelines is given by Shneiderman (1998).

The principle of user familiarity suggests that users should not be forced to adapt to an interface because it is convenient to implement. The interface should use terms familiar to the user and the objects manipulated by the system should be directly related to the user's environment. For example, if a system is designed for use by air traffic controllers, the objects manipulated should be aircraft, flight paths, beacons, etc. Associated operations might be increase or reduce aircraft speed, adjust

Figure 15.3 User interface design principles

Principle	Description
User familiarity	The interface should use terms and concepts which are drawn from the experience of the people who will make most use of the system.
Consistency	The interface should be consistent in that, wherever possible, comparable operations should be activated in the same way.
Minimal surprise	Users should never be surprised by the behaviour of a system.
Recoverability	The interface should include mechanisms to allow users to recover from errors.
User guidance	The interface should provide meaningful feedback when errors occur and provide context-sensitive user help facilities.
User diversity	The interface should provide appropriate interaction facilities for different types of system user.

heading and change height. The underlying implementation of the interface in terms of files and data structures should be hidden from the end-user.

The principle of user interface consistency means that system commands and menus should have the same format, parameters should be passed to all commands in the same way, and command punctuation should be similar. Consistent interfaces reduce user learning time. Knowledge learnt in one command or application is applicable in other parts of the system.

Interface consistency across sub-systems is also important. As far as possible, commands with similar meanings in different sub-systems should be expressed in the same way. Errors are often caused when the same keyboard command, such as 'Control-k' or 'Control-b' means different things in different systems. For example, in the word processor that I used to write this book, 'Control-b' means embolden text but in the graphics program that I used to create diagrams, 'Control-b' means move the selected object behind another object. I make mistakes when using them together and often try to embolden text in a diagram using the key combination. This causes the text to disappear and can be very confusing. Windowing system standards usually define command key shortcuts and should be followed to help avoid this type of error. degree of density

This level of consistency is low-level consistency. Interface designers should always try to achieve this in a user interface. Consistency at a higher level is also sometimes desirable. For example, it may be appropriate to support the same operations (such as print, copy, etc.) on all types of system entities. However, Grudin (1989) points out that complete consistency is neither possible or desirable. It may be sensible to implement deletion from a desktop by dragging entities into a trashcan. It would be unnatural to delete text in a word processor in this way.

The principle of minimal surprise is appropriate because users get very irritated when a system behaves in an unexpected way. As a system is used, users build a mental model of how the system works. If an action in one context causes a particular type of change, it is reasonable to expect that the same action in a different context will cause a comparable change. If something completely different happens, the user is both surprised and confused. Interface designers must therefore ensure that comparable actions have comparable effects.

The principle of recoverability is important because users inevitably make mistakes when using a system. The interface design can minimise these mistakes (e.g. using menus means that typing mistakes are avoided) but mistakes can never be completely eliminated. User interfaces should contain facilities allowing users to recover from their mistakes. These can be of two kinds:

1. *Confirmation of destructive actions* If users specify an action which is potentially destructive, they should be asked to confirm that this is really what they want before any information is destroyed.

2. *The provision of an undo facility* Undo restores the system to a state before the action occurred. Multiple levels of undo are useful as users don't always recognise immediately that a mistake has been made.

A related principle is the principle of user assistance. Interfaces should have built-in user assistance or help facilities. These should be integrated with the system and should provide different levels of help and advice. Levels should range from basic information on getting started with the system to a full description of system facilities. Help facilities, as discussed in section 15.4, should be structured and users should not be overwhelmed with information when they ask for help.

The principle of user diversity recognises that, for many interactive systems, there may be different types of users. Some users will be casual users who interact occasionally with the system while others may be 'power users' who use the system for several hours each day. Casual users need interfaces that provide guidance whereas power users require shortcuts that allow them to interact as quickly as possible. Furthermore, users may suffer from different types of disability and, if possible, the interface should be adaptable to cope with these. Therefore, it may be necessary to provide facilities to display enlarged text, to replace sound with text, to produce very large buttons and so on.

The principle of recognising user diversity can conflict with the other interface design principles because some types of user may prefer to have very rapid interaction rather than, for example, user interface consistency. Similarly, the level of user guidance required can be radically different for different types of user and it may be impossible to develop support that is suitable for all types of user. The interface designer must inevitably make compromises depending on the actual users of the system.

15.2 User interaction

The designer of a user interface to a computer is faced with two key issues. How can information from the user be provided to the computer system and how can information from the computer system be presented to the user? A coherent user interface must integrate user interaction and information presentation.

User interaction is the focus of this section, with information presentation covered in section 15.3. User interaction means issuing commands and associated data to the computer system. On early computers, the only way to do this was through a command-line interface where a special-purpose language was used to communicate with the machine. However, this approach was only usable by experts and a number of other approaches have evolved that are easier to use. Shneiderman (1998) has classified these different forms of interaction into five primary styles:

1. *Direct manipulation* where the user interacts directly with objects on the screen. For example, to delete a file, a user may drag it to a trashcan.
2. *Menu selection* where a user selects a command from a list of possibilities (a menu). It is often the case that another screen object is selected at the same

Interaction style	Main advantages	Main disadvantages	Application examples
Direct manipulation	Fast and intuitive interaction Easy to learn	May be hard to implement Only suitable where there is a visual metaphor for tasks and objects	Video games CAD systems
Menu selection	Avoids user error Little typing required	Slow for experienced users Can become complex if many menu options	Most general-purpose systems
Form fill-in	Simple data entry Easy to learn	Takes up a lot of screen space	Stock control Personal loan processing
Command language	Powerful and flexible	Hard to learn Poor error management	Operating systems Library information retrieval systems
Natural language	Accessible to casual users Easily extended	Requires more typing Natural language understanding systems are unreliable	Timetable systems WWW information retrieval systems

Figure 15.4 Advantages and disadvantages of interaction styles

time and the command operates on that object. In this approach, to delete a file, the user selects the file, then selects the delete command.

3. *Form fill-in* where a user fills in the fields of a form. Some fields may have associated menus and the form may have action 'buttons' that, when pressed, cause some action to be initiated. It would be artificial to delete a file using a form-based interface. It would involve filling in the name of the file, then 'pressing' a delete button.
4. *Command language* where the user issues a special command and associated parameters to instruct the system what to do. To delete a file, the user issues a delete command with the filename as a parameter.
5. *Natural language* where the user issues a command in natural language. To delete a file, the user might therefore type 'delete the file named xxx'.

Each of these different styles of interaction has advantages and disadvantages and are best suited to different types of application and users (Shneiderman, 1998). Figure 15.4 shows the main advantages and disadvantages of these styles and suggests types of application where they might be used.

Of course, these interaction styles may be mixed and several different styles are used in the same application. For example, Microsoft Windows supports direct manipulation of the iconic representation of files and directories, menu-based command selection and, for some commands such as configuration commands, the user must fill in a special-purpose form that is presented to them.

User interfaces on the World Wide Web are based on the support provided by HTML (the page description language used for web pages) along with languages

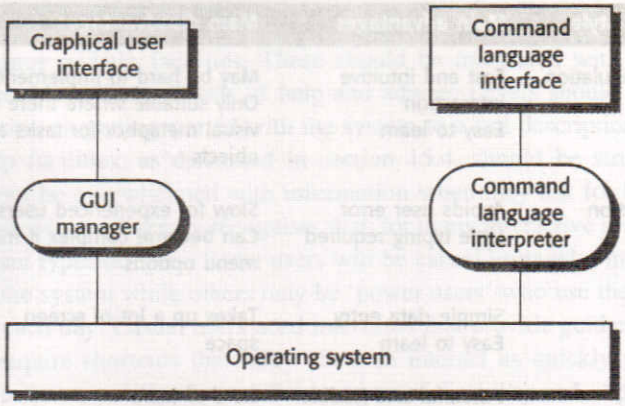


Figure 15.5 Multiple user interfaces

such as Java that can associate programs with components on a page. As these web-based interfaces are often designed for casual users, they mostly use forms-based interfaces. It is possible to construct direct manipulation interfaces on the web but, at the time of writing, this is still a complex programming task.

In principle, it should be possible to separate the interaction style from the underlying entities that are manipulated through the user interface. This was the basis of the Seeheim model (Pfaff and ten Hagen, 1985) of user interface management. In this model, the presentation of information, the dialogue management and the application are separate. In reality, this model is an ideal one rather than a practical one but it is certainly possible to have separate interfaces for different classes of users (casual users and experienced users, say) that interact with the same underlying system. This is illustrated in Figure 15.5 which shows a command language interface and a graphical interface to an underlying operating system such as Linux.

This separation of presentation, interaction and the entities that are involved in the user interface is also fundamental to the Model-View-Controller approach that I discuss in the next section. This is comparable to the Seeheim model but is used to implement the user interface to objects rather than entire applications.

15.3 Information presentation

All interactive systems have to provide some way of presenting information to users. The information presentation may simply be a direct representation of the input information (e.g. text in a word processor) or it may present the information graphically. It is good system design practice to keep the software required for information presentation separate from the information itself. To some extent, this contradicts object-oriented philosophy which suggests that operations on data

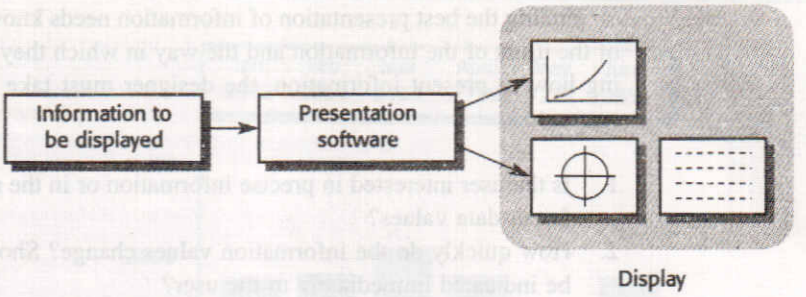


Figure 15.6 Information presentation

should be defined with the data itself. However, this presupposes that the designer of the objects always knows the best way to present information; this is definitely not always true. It is often difficult to know the best way to present data when it is being defined and object structures should not 'hard-wire' presentation operations.

By separating the presentation system from the data, the representation on the user's screen can be changed without having to change the underlying computational system. This is illustrated in Figure 15.6.

The MVC approach (Figure 15.7), first made widely available in Smalltalk (Goldberg and Robson, 1983), is an effective way to support multiple presentations of data. Users can interact with each presentation using a style that is appropriate to it. The data to be displayed is encapsulated in a model object. Each model object may have a number of separate view objects associated with it where each view is a different display representation of the model. I have already illustrated this in the previous chapter where I discussed the MVC approach as an object-oriented framework.

Each view has an associated controller object that handles user input and device interaction. Therefore, a model that represents numeric data may have a view that represents the data as a histogram and a view that presents the data as a table. The model may be edited by changing the values in the table or by lengthening or shortening the bars in the histogram. I have illustrated this in Chapter 14 (Figure 14.13) where I discussed the use of the Observer pattern in implementing the MVC framework.

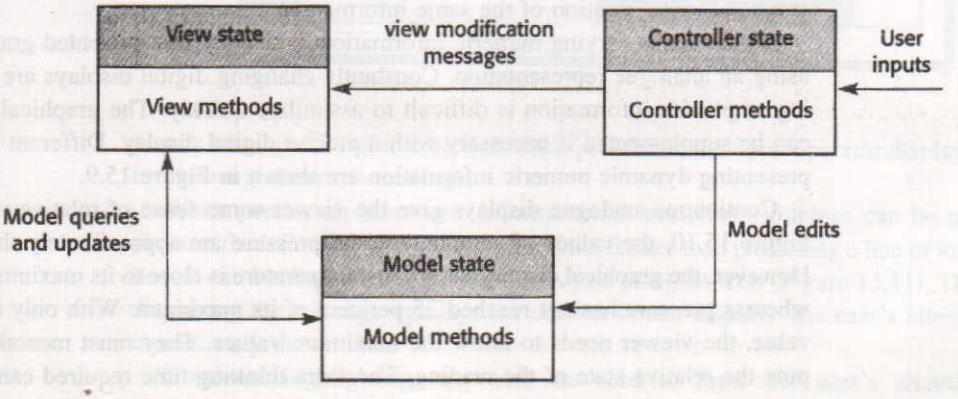


Figure 15.7 The MVC model of user interaction

Finding the best presentation of information needs knowledge of the background of the users of the information and the way in which they use the system. In deciding how to present information, the designer must take a number of factors into account:

1. Is the user interested in precise information or in the relationships between different data values?
2. How quickly do the information values change? Should the change in a value be indicated immediately to the user?
3. Must the user take some action in response to a change in information?
4. Does the user need to interact with the displayed information via a direct manipulation interface?
5. Is the information to be displayed textual or numeric? Are relative values of information items important?

Information that does not change during a session may be presented either graphically or as text depending on the application. Textual presentation takes up less screen space but cannot be read 'at a glance'. Information that does not change should be distinguished from dynamic information by using a different presentation style. For example, all static information could be presented in a particular font, or could be highlighted using a particular colour or could always have an associated icon.

Information should be represented as text when precise numeric information is required and the information changes relatively slowly. If the data changes quickly or if the relationships between data are significant, graphical presentation should usually be used.

For example, consider a system which records and summarises the sales figures for a company on a monthly basis. Figure 15.8 illustrates how the same information can be presented as text or in a graphical form.

Managers studying sales figures are usually more interested in trends or anomalous figures rather than precise values. Graphical presentation of this information, as a histogram, makes the anomalous figures in March and May stand out from the others. Figure 15.8 also illustrates how textual presentation takes less space than a graphical representation of the same information.

Dynamically varying numeric information is usually best presented graphically using an analogue representation. Constantly changing digital displays are confusing as precise information is difficult to assimilate quickly. The graphical display can be supplemented if necessary with a precise digital display. Different ways of presenting dynamic numeric information are shown in Figure 15.9.

Continuous analogue displays give the viewer some sense of relative value. In Figure 15.10, the values of temperature and pressure are approximately the same. However, the graphical display shows that temperature is close to its maximum value whereas pressure has not reached 25 per cent of its maximum. With only a digital value, the viewer needs to know the maximum values. They must mentally compute the relative state of the reading. The extra thinking time required can lead to

Figure 15.8 Alternative information presentations

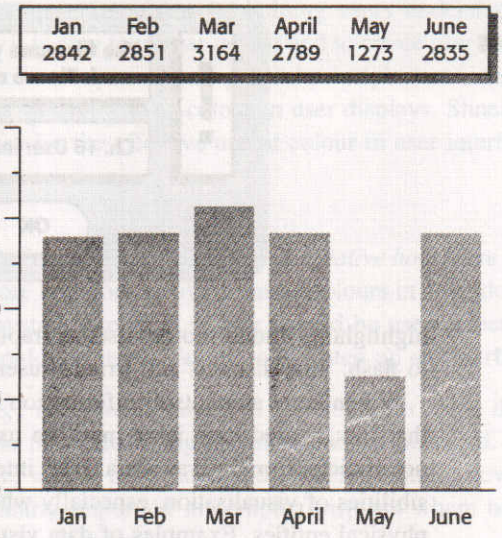


Figure 15.9 Methods of presenting dynamically varying numeric information

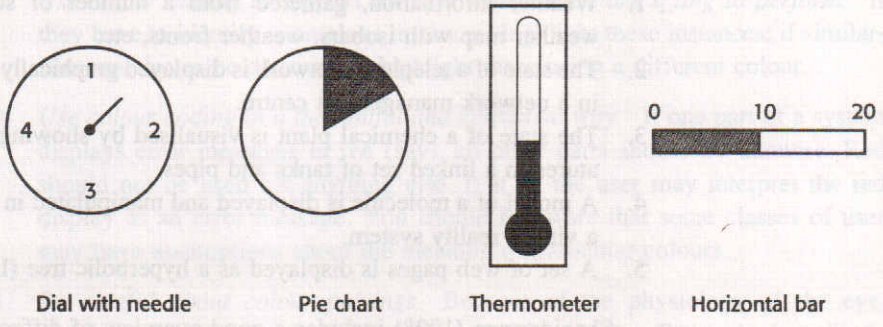
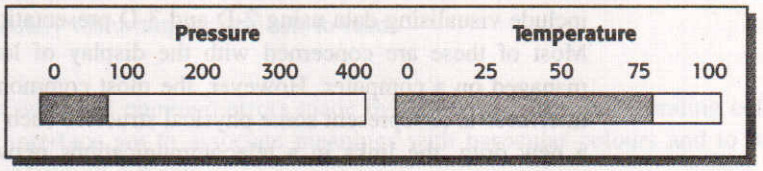


Figure 15.10 Graphical information display showing relative values



human errors in stressful situations when problems occur and operator displays may be showing abnormal readings.

When precise alphanumeric information is presented, graphics can be used to pick out the information from its background. Rather than presenting a line of information, it can be displayed in a box or indicated using an icon (Figure 15.11). The box displaying the message overlays the current screen display. The user's attention is immediately drawn to it.

Graphical highlighting may also be used to draw the user's attention to changes to parts of the display. However, if these changes occur rapidly, graphical

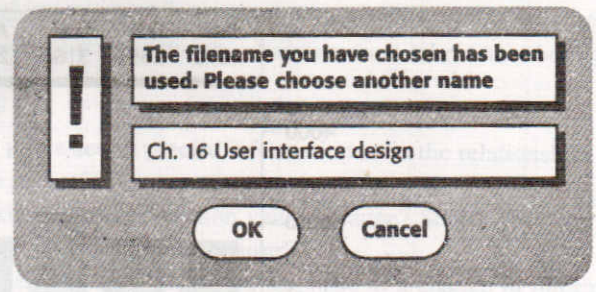


Figure 15.11
Textual highlighting of alphanumeric information

highlighting should not be used as rapid changes may cause the display to appear to flash. This distracts and irritates users.

When large amounts of information have to be presented, abstract visualisations that link related data items may be used. This can expose relationships that are not obvious from the raw data. User interface designers should be aware of the possibilities of visualisation, especially when the system user interface must represent physical entities. Examples of data visualisations are:

1. Weather information, gathered from a number of sources, is shown as a weather map with isobars, weather fronts, etc.
2. The state of a telephone network is displayed graphically as a linked set of nodes in a network management centre.
3. The state of a chemical plant is visualised by showing pressures and temperatures in a linked set of tanks and pipes.
4. A model of a molecule is displayed and manipulated in three dimensions using a virtual reality system.
5. A set of web pages is displayed as a hyperbolic tree (Lamping *et al.*, 1995).

Shneiderman (1998) includes a good overview of different approaches to visualisation and identifies different classes of visualisation that may be used. These include visualising data using 2-D and 3-D presentations and as trees or networks. Most of these are concerned with the display of large amounts of information managed on a computer. However, the most common use of visualisation in user interfaces is to represent some physical structure such as the molecular structure of a new drug, the links in a telecommunications network, etc. Three-dimensional presentations that may use special virtual reality equipment are particularly effective in product visualisations. Direct manipulation of these visualisations is a very effective way to interact with the data.

15.3.1 Colour in interface design

All interactive systems, apart from specialised, small-screen systems, support colour displays and user interfaces make use of colour in different ways. In some systems (such as word processors) it is simply used for highlighting; in others (such as CAD systems) it is used to show different layers in a design.

Colour can improve user interfaces by helping users understand and manage complexity. However, it is easy to misuse colour and to create user interfaces that are visually unattractive and error-prone. As a general principle, user interface designers should be conservative in their use of colour in user displays. Shneiderman (1998) gives 14 key guidelines for the effective use of colour in user interfaces. The most important of these are:

1. *Limit the number of colours used and be conservative how these are used* You should not use more than four or five separate colours in a window and no more than seven in a system interface. Colour should be used selectively and consistently. You should not use it simply to brighten up an interface.

2. *Use colour change to show a change in system status* If a display changes colour, this should mean that a significant event has occurred. Thus, in a fuel gauge, a change of colour may indicate that fuel is running low. Colour highlighting is particularly important in complex displays where hundreds of distinct entities may be displayed.

3. *Use colour coding to support the task which users are trying to perform* If they have to identify anomalous instances, highlight these instances; if similarities are also to be discovered, highlight these using a different colour.

4. *Use colour coding in a thoughtful and consistent way* If one part of a system displays error messages in red (say), all other parts should do likewise. Red should not be used for anything else. If it is, the user may interpret the red display as an error message. You should be aware that some classes of user may have assumptions about the meaning of particular colours.

5. *Be careful about colour pairings* Because of the physiology of the eye, people cannot focus on red and blue simultaneously. Eyestrain is a likely consequence of a red on blue display. Other colour combinations may also be visually disturbing or difficult to read.

The two most common errors made by designers when incorporating colour in a user interface are to associate meanings with particular colours and to use too many colours in a display.

Colour should not be used to represent meaning, for two reasons. About 10 per cent of men are colour-blind and may misinterpret the meaning. Human colour perceptions are different and there are different conventions in different professions about the meaning of particular colours. Users with different backgrounds may unconsciously interpret the same colour in different ways. For example, to a driver red usually means *danger*. However, to a chemist, red means *hot*.

If too many colours are used or if the colours are too bright, the display may be confusing. The mass of colour may disturb the user (in the same way that some abstract paintings cannot be viewed comfortably for a long time) and cause visual fatigue. User confusion is also possible if colours are used inconsistently.

15.4 User support

A design principle suggested in the first section of this chapter was that user interfaces should always provide some form of online help system. Help systems are one facet of a general part of user interface design, namely the provision of user guidance which covers three areas:

- the messages produced by the system in response to user actions;
- the online help system;
- the documentation provided with the system.

The design of useful and informative information for users should be taken seriously and should be subject to the same quality process as designs or programs. Managers must allow sufficient time and effort for message design and it may be appropriate to involve professional writers and graphic artists in the process. When designing error messages or help text, the factors shown in Figure 15.12 should be taken into account.

15.4.1 Error messages

The first impression that users may have of a software system is the system error messages. Inexperienced users may start work, make an initial error and immedi-

Figure 15.12 Design factors in message wording

Factor	Description
Context	The user guidance system should be aware of what the user is doing and should adjust the output message to the current context.
Experience	As users become familiar with a system they become irritated by long, 'meaningful' messages. However, beginners find it difficult to understand short terse statements of the problem. The user guidance system should provide both types of message and allow the user to control message conciseness.
Skill level	Messages should be tailored to the users' skills as well as their experience. Messages for the different classes of user may be expressed in different ways depending on the terminology which is familiar to the reader.
Style	Messages should be positive rather than negative. They should use the active rather than the passive mode of address. They should never be insulting or try to be funny.
Culture	Wherever possible, the designer of messages should be familiar with the culture of the country where the system is sold. There are distinct cultural differences between Europe, Asia and America. A suitable message for one culture might be unacceptable in another.

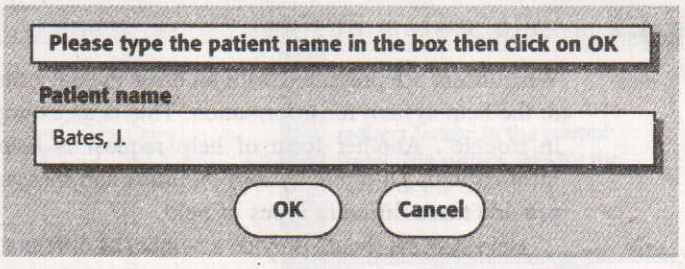


Figure 15.13
Nurse input of a patient's name

ately have to understand the resulting error message. This can be difficult enough for skilled software engineers. It is often impossible for inexperienced or casual system users.

The background and experience of users should be anticipated when designing error messages. For example, say a system user is a nurse in an intensive-care ward in a hospital. Patient monitoring is carried out by a computer system. To view a patient's current state (heart rate, temperature, etc.), the system user selects 'display' from a menu and inputs the patient's name in the box as shown in Figure 15.13.

In this case, say the patient's name was Pates rather than Bates so that the nar input by the nurse could not be recognised. The system generates an error message. Error messages should always be polite, concise, consistent and constructive. They must not be abusive and should not have associated beeps or other noises which might embarrass the user. Wherever possible, the message should suggest how the error might be corrected. The error message should be linked to a context-sensitive online help system.

Figure 15.14 shows examples of good and bad error messages. The left-hand message is badly designed. It is negative (it accuses the user of making an error), it is not tailored to the user's skill and experience level and it does not take context information into account. It does not suggest how the situation might be rectified. It uses system-specific terms (patient-id) rather than user-oriented language. The right-hand message is better. It is positive, implying that the problem is a system rather than a user problem. It identifies the problem in the nurse's terms, and offers an easy way to correct the mistake by pressing a single button. The help system is available if required.

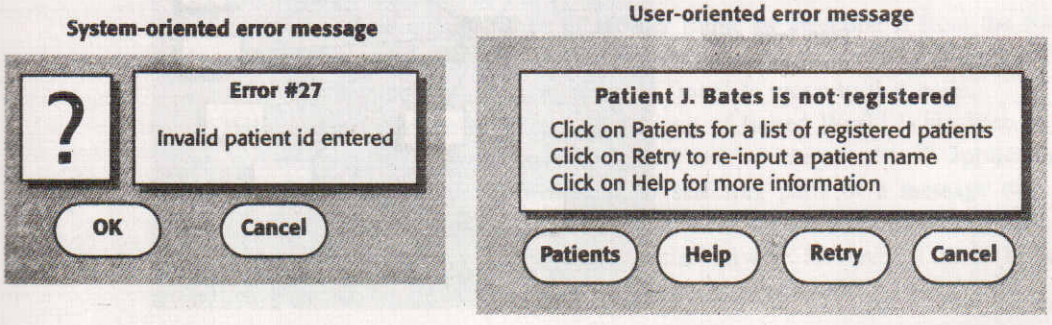


Figure 15.14 System and user-oriented error messages

15.4.2 Help system design

When users are presented with an error message they do not understand, they turn to the help system for information. This is an example of *help!* meaning 'help, I'm in trouble'. Another form of help request is *help?* which means 'help, I want information'. Different system facilities and message structures may be needed to provide these different types of help.

Help systems should provide a number of different user entry points (Figure 15.15). These should allow the user to enter the help system at the top of the message hierarchy and browse for information. Alternatively, they may enter the help system to get an explanation of an error message or may request an explanation of a particular application command.

All comprehensive help systems have a complex network structure where each frame of help information may refer to several other information frames. The structure of this network is usually hierarchical with cross-links as shown in Figure 15.15. General information is held at the top of the hierarchy, detailed information at the bottom.

Problems can arise with help systems when users enter the network after making a mistake and then navigate around the network. Within a short time, they can become hopelessly lost. They must abandon the session and start again at some known point in the network.

Displaying help information in multiple windows can help alleviate this situation. Figure 15.16 shows a screen display where there are three help windows.

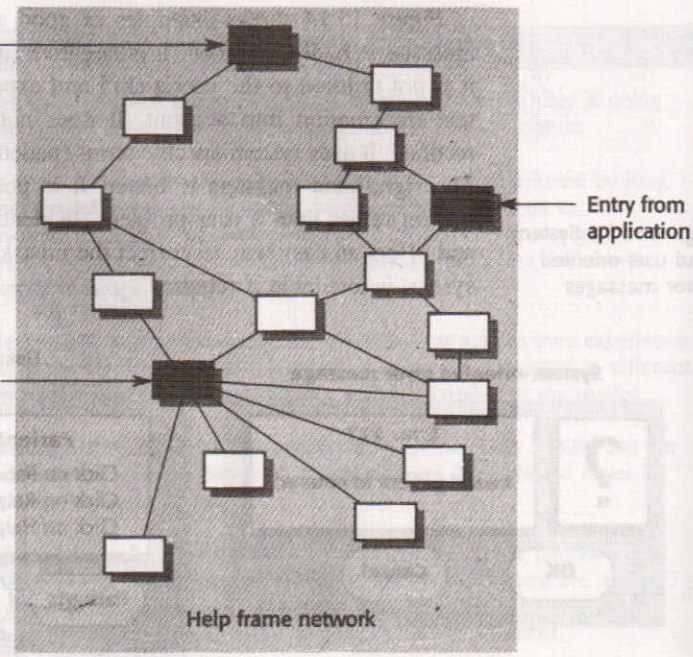


Figure 15.15 Entry points to a help system

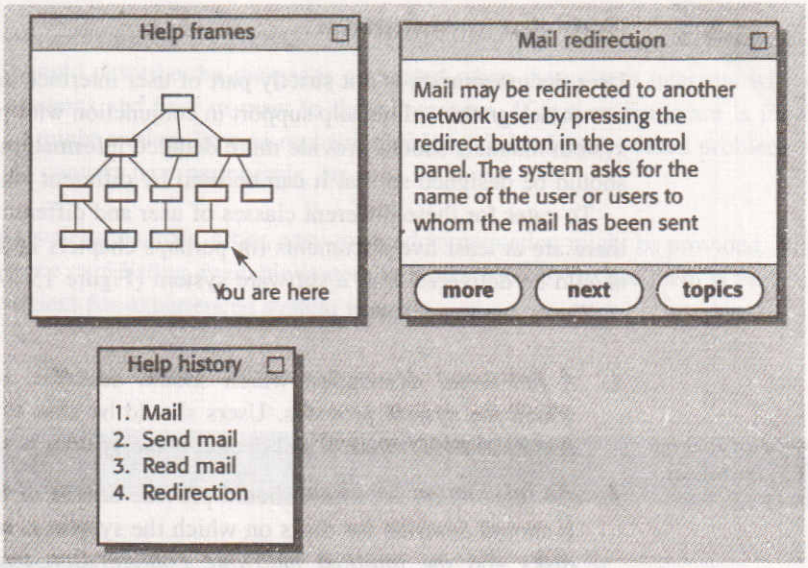


Figure 15.16 Help system windows

However, screen space is always limited and the designer must be aware that displaying extra windows may obscure other information which is important.

The text in a help system should be prepared with the help of application specialists. The help frame should not simply be a reproduction of the user manual as people read paper and screens in different ways. The text itself, its layout and style have to be carefully signed to ensure that it is readable in a relatively small window. In Figure 15.16, the help frame ('Mail redirection') is relatively short. You should not overwhelm the user with information in any one frame. Three buttons are provided in the help frame to request more information, to move on to the next help frame and to call up a list of topics on which help is available.

The 'history' window shows the frames which have already been visited. It should be possible to return to these frames by picking an item from this list. The navigation window is a graphical 'map' of the help system network. The current position in this map should be highlighted by using colour, shading or, in this case, by annotation.

Users should be able to move to another frame by selecting it from the frame being read, by selecting a frame in the 'history window' to reread or to retrace their steps or by selecting a node in the network 'map' to move to that node.

Help systems may be implemented as a set of linked World Wide Web pages or by using a generic hypertext system that can be integrated with applications. The hierarchy may be easily traversed by selecting parts of a message that are highlighted as links. World Wide Web systems have the advantage that they are easy to implement and do not require any special software. However, it can be difficult to link these to the application to provide context-sensitive help.

15.4.3 User documentation

User documentation is not strictly part of user interface design but it is good practice to design the online help support in conjunction with paper documentation. The system manuals should provide more detailed information than the online help and should be designed so that it can be used by different classes of system end-user.

To cater for these different classes of user and different levels of user expertise, there are at least five documents (or perhaps chapters in a single document) which should be delivered with a software system (Figure 15.17).

These documents are:

1. *A functional description* which should describe, very briefly, the services which the system provides. Users should be able to read this document with an introductory manual and decide if the system is what they need.
2. *An installation document* should provide details of how to install the system. It should describe the disks on which the system is supplied, the files on these disks and the minimal hardware configuration required. It should include installation instructions and details of how to set up configuration-dependent files.
3. *An introductory manual* which should present an informal introduction to the system, describing its 'normal' usage. It should describe how to get started and how end-users might make use of the common system facilities. It should be liberally illustrated with examples. It should include information on how to recover from mistakes and restart useful work. Carroll (1992) suggests that focusing on error recovery and minimising the information that users have to read by eliminating redundancy is an effective way to organise introductory manuals.
4. *A reference manual* should describe the system facilities and their usage, provide a list of error messages and possible causes and describe how to recover from detected errors.

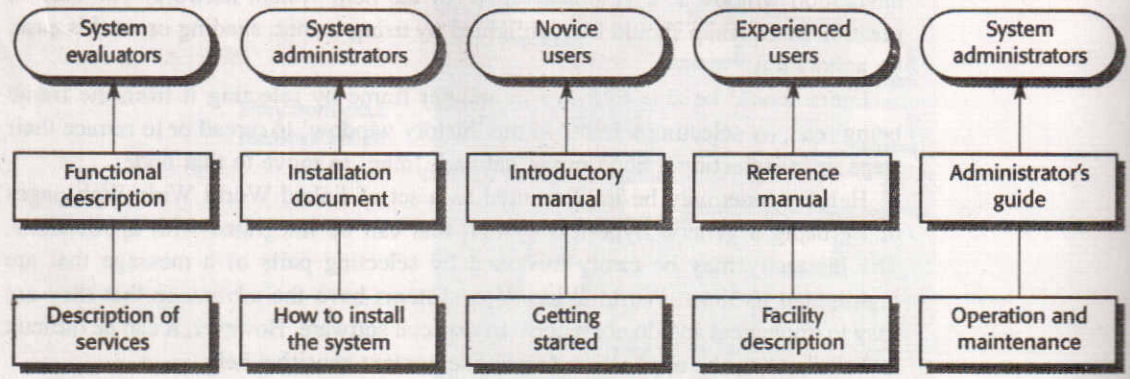


Figure 15.17
Types of document produced to support users

5. *An administrator's manual* may be provided for some types of system. This should describe the messages generated when the system interacts with other systems and how to react to these messages. If system hardware is involved, it might explain how to recognise and repair hardware-related problems, how to connect new peripherals, etc.

As well as manuals, other, easy-to-use documentation might be provided. A quick reference card listing available system facilities and how to use them is particularly convenient for experienced system users.

15.5 Interface evaluation

Interface evaluation is the process of assessing the usability of an interface and checking that it meets user requirements. Therefore, it should be part of the normal verification and validation process for software systems. Neilsen (1993) in his book on usability engineering includes a good chapter on this topic.

Ideally, an evaluation should be conducted against a usability specification based on usability attributes as shown in Figure 15.18. Metrics for these usability attributes can be devised. For example, a learnability metric might state that an operator who is familiar with the work supported should be able to use 80 per cent of the system functionality after a three-hour training session. However, it is more common to specify usability (if it is specified at all) qualitatively rather than using metrics. Interface designers therefore have to use their judgement and experience in the interface evaluation.

Systematic evaluation of a user interface design can be an expensive process involving cognitive scientists and graphics designers. It may involve designing and carrying out a statistically significant number of experiments with typical users in

Figure 15.18
Usability attributes

Attribute	Description
Learnability	How long does it take a new user to become productive with the system?
Speed of operation	How well does the system response match the user's work practice?
Robustness	How tolerant is the system of user error?
Recoverability	How good is the system at recovering from user errors?
Adaptability	How closely is the system tied to a single model of work?

specially constructed laboratories fitted with monitoring equipment. A user interface evaluation of this kind is economically unrealistic for systems developed by small organisations with limited resources.

There are a number of simpler, less expensive techniques of user interface evaluation which can identify particular user interface design deficiencies:

1. questionnaires which collect information about what users thought of the interface;
2. observation of users at work with the system and 'thinking aloud' about how they are trying to use the system to accomplish some task;
3. video 'snapshots' of typical system use;
4. the inclusion in the software of code which collects information about the most-used facilities and the most common errors.

Surveying users by using a questionnaire is a relatively cheap way of evaluating an interface. The questions should be precise rather than general. It is no use asking questions like 'Please comment on the usability of the interface' as the responses will probably vary so much that no common trend will emerge. Rather, specific questions such as 'Please rate the understandability of the error messages on a scale from 1 to 5. A rating of 1 means very clear and 5 means incomprehensible' are better. They are both easier to answer and more likely to provide useful information to improve the interface.

Users should be asked to rate their own experience and background when filling in the questionnaire. This allows the designer to find out if users from any particular background have problems with the interface. Questionnaires can even be used before any executable system is available if a paper mock-up of the interface is constructed and evaluated.

Observation-based evaluation simply involves watching users as they use a system, looking at the facilities used, the errors made, etc. This can be supplemented by 'think aloud' sessions where users talk about what they are trying to achieve, how they understand the system and how they are trying to use the system to accomplish their objectives.

Relatively low-cost video equipment means that direct observation can be supported by recording user sessions for later analysis. Complete video analysis is expensive and requires a specially equipped evaluation suite with several cameras focused on the user and on the screen. However, video recording of selected user operations can be helpful to detect problems. Other evaluation methods must be used to find out which operations cause user difficulties.

Analysis of recordings allows the designer to find out if the interface requires too much hand movement (a problem with some systems is that users must regularly move their hands from keyboard to mouse) and to see if unnatural eye movements are necessary. An interface which requires many shifts of focus may mean that the user makes more errors and misses parts of the display.

Instrumenting code to collect usage statistics allows interfaces to be improved in a number of ways. The most common operations can be detected. The interface

can be reorganised so that these are the fastest to select. For example, if pop-up or pull-down menus are used, the most frequent operations should be at the top of the menu and destructive operations towards the bottom. Code instrumentation also allows error-prone commands to be detected and modified.

Finally, a means of easy user response can be provided by equipping each program with a 'gripe' command which the user can use to pass messages to the tool maintainer. This makes users feel that their views are being considered. The interface designer and other engineers can gain rapid feedback about individual problems.

None of these relatively simple approaches to user interface evaluation is foolproof and they are unlikely to detect all user interface problems. However, the techniques can be used with a group of volunteers before a system is released without a large outlay of resources. Many of the worst problems of the user interface design can then be discovered and corrected.

KEY POINTS

- The interface design process should be user-centred. An interface should interact with users in their terms, should be logical and consistent and should include facilities to help users with the system and to recover from their mistakes.
- Styles of interaction with a software system include direct manipulation, menu systems, form fill-in, command languages and natural language.
- Graphical information display should be used when it is intended to present trends and approximate values. Digital display should only be used when precision is required.
- Colour should be used sparingly and consistently in user interfaces. Designers should take account of the fact that a significant number of people are colour-blind.
- User help systems should provide two kinds of help: Help! which is 'help, I'm in trouble' and Help? which is 'help, I need information .
- Error messages should not suggest that the user is to blame. They should offer suggestions on how to repair the error and provide a link to a help system.
- User documentation should include beginner's and reference manuals. Separate documents for system administrators should be provided.
- The system specification should include, wherever possible, quantitative values for usability attributes and the evaluation process should check the system against these requirements.

FURTHER READING

Human-Computer Interaction, 2nd edition. A good general text whose strengths are a focus on design issues and cooperative work. (A. Dix, J. Finlay, G. Abowd and R. Beale, 1998, Prentice-Hall.)

Designing the User Interface. This is the latest revision of what was probably the first textbook in this area. An excellent, wide-ranging book. (B. Shneiderman, 1998, Addison-Wesley.)

Developing User Interfaces. This book focuses on the implementation rather than on the design of user interfaces. It is therefore a good complement to other texts that concentrate on interface design. (D. Olsen, 1998, Morgan Kaufmann.)

EXERCISES

15.1 I suggested in section 15.1 that the objects manipulated by users should be drawn from their domain rather than a computer domain. Suggest appropriate objects for the following types of user and system:

- a warehouse assistant using an automated parts catalogue;
- an airline pilot using an aircraft safety monitoring system;
- a manager manipulating a financial database;
- a policeman using a patrol car control system.

15.2 Suggest situations where it is unwise or impossible to provide a consistent user interface.

15.3 What factors have to be taken into account when designing a menu-based interface for 'walk-up' systems such as bank ATM machines? Write a critical commentary on the interface of an ATM that you use.

15.4 Suggest ways in which the user interface to an e-commerce system such as an online bookstore or music retailer might be adapted for users who are physically challenged with some form of visual impairment or problems with muscular control.

15.5 Discuss the advantages of graphical information display and suggest four applications where it would be more appropriate to use graphical rather than digital displays of numeric information.

15.6 What are the guidelines which should be followed when using colour in a user interface? Suggest how colour might be used more effectively in the interface of an application system that you use.

15.7 Write a short set of guidelines for designers of user guidance systems.

15.8 Consider the error messages produced by MS-Windows, Unix, MacOS or some other operating system. Suggest how these might be improved.

15.9 Design a questionnaire to gather information about the user interface of some tool (such as a word processor) with which you are familiar. If possible, distribute this questionnaire to a number of users and try to evaluate the results. What do these tell you about the user interface design?

15.10 Discuss whether it is ethical to instrument software without telling end-users that their work is being monitored.

15.11 What ethical issues might user interface designers face when trying to reconcile the needs of end-users of a system with the needs of the organisation which is paying for the system to be developed?