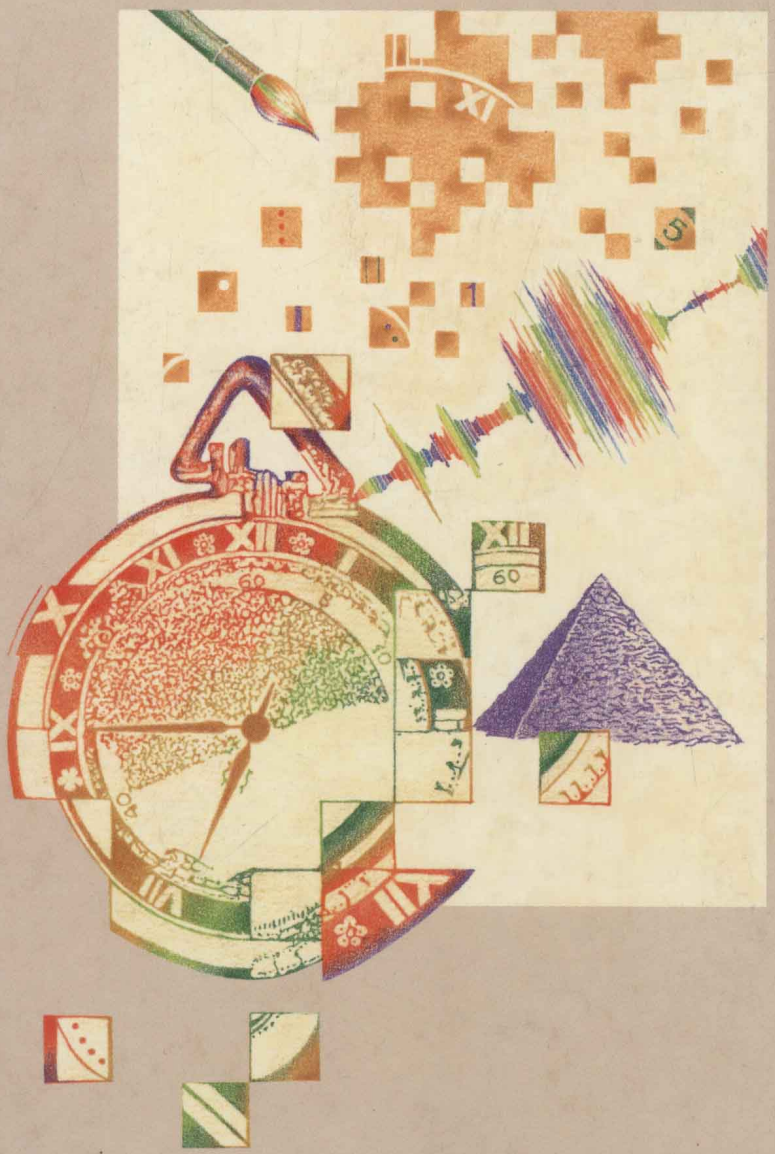
# USER INTERFACE DESIGN



Kevin Cox • David Walker

SECOND EDITION

First Published 1993 by
Prentice Hall
Simon & Schuster Asia Pte Ltd
Alexandra Distripark
Block 4 #04-31
Pasir Panjang Road
Singapore 0511



Printed in Singapore

1 2 3 4 5 97 96 95 94 93

ISBN 0-13-952888-1

*Cover design by Viscom Design Associates*

The first edition of this book was published
by Advanced Education Software, July 1990

Prentice-Hall International (UK) Limited, *London*
Prentice-Hall of Australia Pty. Limited, *Sydney*
Prentice-Hall Canada Inc., *Toronto*
Prentice-Hall Hispanoamericana, S.A., *Mexico*
Prentice-Hall of India Private Limited, *New Delhi*
Prentice-Hall of Japan, Inc., *Tokyo*
Editora Prentice-Hall do Brasil, Ltda., *Rio de Janeiro*
Prentice-Hall, Inc., *Englewood Cliffs, New Jersey*

# USER-INTERFACE DESIGN

# Preface

We decided to write this book because we became tired of waiting for someone else to do it. Many of us who design and construct computer systems and those of us who try to teach others how to do it are unhappy with the standard text books on system design. They have good information on techniques to help us understand and control our computing system development, but they are of little help in designing good computer systems. We know that there is more to making a computer system than investigating things, drawing some data flow diagrams, data modelling, writing and testing programs, and implementing the finished system. We know that making good computer systems is like all interesting human activities: it is a creative, stimulating, difficult task punctuated with flashes of insight, recognition of patterns, and emotional satisfaction of a job well done. We know that it is a very human experience spiced with marvellous technical gadgetry, randomness, variety and uncertainty.

You cannot create a computer system by applying a formula. There is no magic button to press and a machine will produce a final product. There is a prevailing myth that a technological solution exists that will allow us to create great computer systems at the press of a button, or by the application of a rigorous proven recipe. We search for the ultimate CASE tool to solve the backlogs of design and construction. These attempts are doomed to failure as there is no prescription. The attempts may help us make better systems but they do not help us design, because design is not prescriptive. Even the Chinese language helps preserve the myth as the word for computer can be translated as electric brain.

Other professions know that making human artefacts takes leaps of imagination. They know that we think visually as well as analytically. They know that the person is more important than the machine. Computing professionals know this as well, but the nature of the discipline and the nature of the technology has made us think that there has to be a computer program that will design computer systems. After all we try to make computer programs to design in other professional areas, why not our own. As a profession we have been afraid to recognise, acknowledge, accept and use our humanness. We have

wanted to become mechanical in our design. We have wanted our techniques to reflect the artefacts with which we work.

In this book we start from a different paradigm for the design of computer systems. We think of the design of computer systems as designing something with which we will communicate with other humans. When we make a new computer system we have made another way to express ourselves and our ideas to another person. All computing systems are built for human purposes, all computing systems fit into a world of human needs. We do not create computing systems to talk to other computer systems. Even if they do communicate with one another they do so because we want them to for our own purposes. Users of computer systems communicate by proxy with designers and programmers. When you write a computer program you do not write it to communicate with a machine; you write it to communicate with a person.

When we take this view of our job as designers we can see that the focus of most of our traditional computing training is misdirected. If you look at many curricula for computing professionals you see it weighted with such things as mathematics, computer programming, data modelling and hardware constructs. You occasionally see a little on human psychology or writing skills but you never see an orientation towards human communication. You see courses on computer systems design which only mention users in passing. Students spend all their time communicating with machines using mechanistic principles. No wonder we get *unfriendly* computer systems. They were designed for other computers, not for us.

We have attempted to redress the balance and to put people back into the centre of design. We propose to design for humans as humans. We are not electric brains, we do not want to be electric brains, and we refuse to act as though we are. In this book you will find much more on sketching than you will on mathematical proofs. You will find a chapter on testing with people and a passing reference to testing programs. You will find a lot on writing manuals and a little on program structure. You will see how to design artefacts that people can understand and you will find little on program structure design.

Writing computer systems is a fun activity. It is fun because of the human component. The education industry is rapidly destroying the fun and replacing it with sterility. We would hope that this book might reverse the trend of people viewing computing professionals as boring, mechanical, electric brains. We hope that more artists, communicators, psychologists and humanists will be attracted to the profession and realise the potential of this most marvellous of human artefacts. We hope that students in the profession will rise above the weight of electrons and communicate with us via their programs.

The book starts with our view of what makes a good computer system. Naturally we believe a good computer system is one that works for and with people. We then discuss the nature of design and how we can design around people. We discuss the way people use computing products and how they think about them. The next chapter describes how we can test our theories and models and show that our products are usable. The rest of the book elaborates on these themes and discusses the object/action paradigm for de-

sign, the use of documentation as an explanation tool, and how we can organise and control developments when we design this way.

We constructed a small computer system (Crossword Designer) to accompany the book. You do not need the program to understand the ideas in the book as we describe the important parts of the program interface within the text. However, using it will make some things clearer. The compromises and mistakes of a real artefact are exposed. The package with all its faults gives a human dimension to the text and you can see how the ideas we espouse are realised. You can obtain a copy of Crossword Designer by contacting the authors of this book at the Computer Department of the City Polytechnic of Hong Kong.

The book was designed to support courses with such titles as *Human-Computer Interface Design* or *Design and Construction of Computer Systems*. The approach is more engineering than psychological and more practical than theoretical. We have found that students want and need guidance in how to approach the design task and want to know what they should do when designing. The text has been used as the basis for semester courses for second-year undergraduate students at both the University of Canberra and the City Polytechnic of Hong Kong and for various industrial short courses. The courses were all supported by practical projects. Usability projects involved testing commonly available software. Design projects have come from areas of topical interest to students and we select new subjects each time we run a course. For illustration and exercises we use any commonly available computer system. Our main usability criterion for software and computer systems has been student ease of access. We have applied the ideas in the book to the design of many interactive computer applications and implemented them on mainframes, mini-computers and personal computers.

During the preparation of this book we have received ideas and help from many people. The University of Canberra supported one of us with leave from teaching. Our colleagues, family and friends have encouraged us. Our many tutors in courses over the years have contributed much and we would like to thank all the students who have patiently tried out the ideas and given us valuable feedback. At the risk of offending everyone we mention no-one in particular.

Most of the ideas in the book can be found in the bibliography. We have referenced material where we know the original source of the idea or we have used similar material to the reference. Our purpose has been to integrate and present ideas in an easily understood and easily applied manner.

Kevin Cox - Hong Kong (CSCOXK@CPHKVX.BITNET)
David Walker - Canberra
March 1992

# Table of Contents

# Chapter 1

# What Makes a Good Computer System?

A characteristic feature of human beings is that they are users of tools. We build things using hammers, chisels and saws, we transport ourselves between two places in cars, trains and aeroplanes, we cook our meals on a stove. A computer is a tool. We use computers to do things such as:

- Control a microwave oven.
- Send the electricity bill for the house with that oven.
- Make models of the atmosphere which predict the changes in world climate caused by the use of that (and many other) ovens.

Use of a tool involves co-operation between a person (the *user*) and the tool. We do not say "The hammer banged in the nail", we say "I banged in the nail using the hammer". The user is in charge. The hammer does not say which nail to put in, or beep at you when you use a large nail rather than a small one. Similarly, the computer does not decide to perform a task, although it is capable of obeying a complex series of instructions. A person must set and switch on the microwave oven, decide that electricity bills have to be sent, and build the atmospheric model.

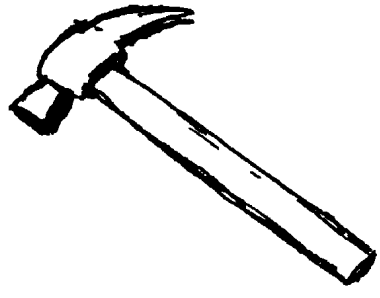
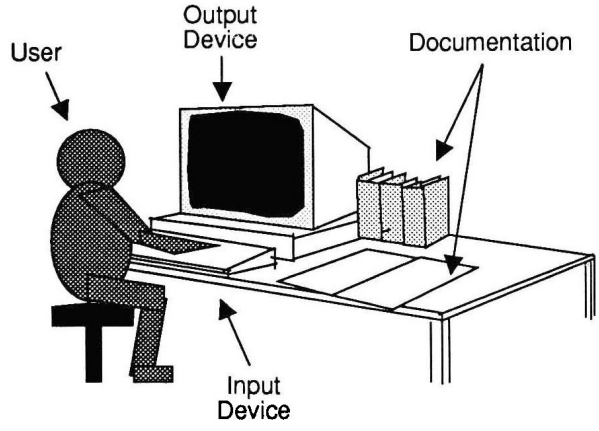
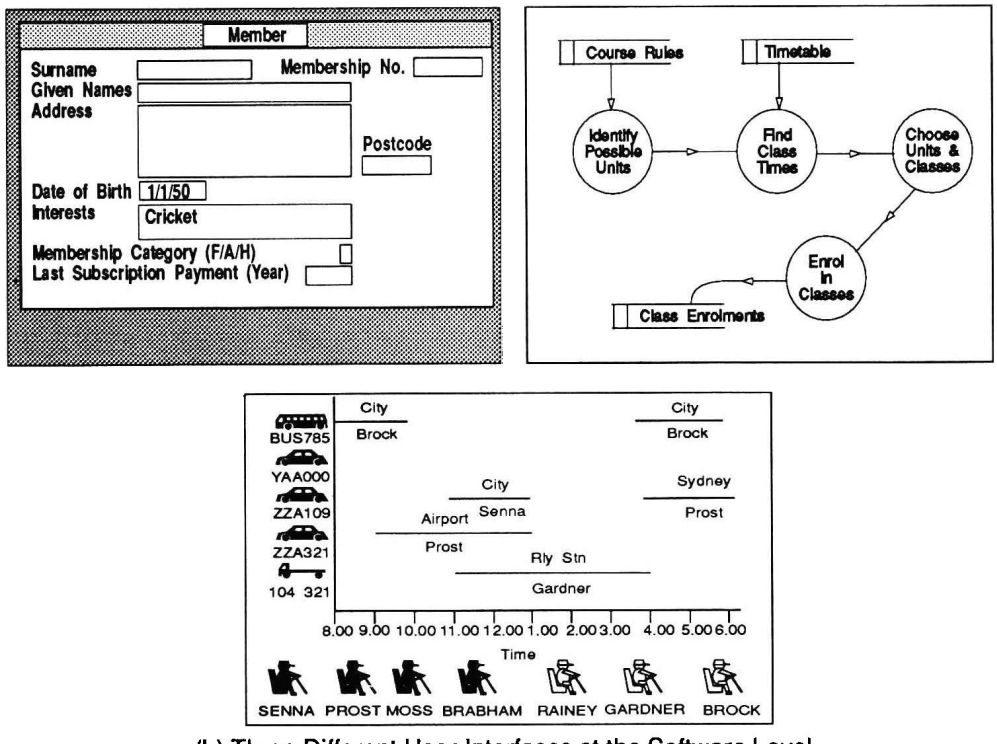


**Figure 1.1** A tool.

Because of this requirement for co-operation, the way in which the person has




(a) The User-Interface at the Hardware Level




(b) Three Different User-Interfaces at the Software Level

**Figure 1.2** The user-interface.

to interact with the tool determines whether or not the tool is usable.

The handle of a hammer must be of an appropriate size, shape and texture for a human hand to grip firmly and must be placed appropriately in relation to the head to provide the correct leverage. The hammer as a whole must not be too heavy to lift, but must be heavy enough to drive the nail home. The user must be able to see where the head will strike in relation to the nail, in order to line up a hit. In addition, the user must have some concept of what a hammer is and what it is used for, or they might believe that it is for stirring soup.

These features constitute the *user-interface* of the hammer. It is easy to imagine a hammer with a poor user-interface: a short, fat, greasy handle, and a ball-shaped head with a cowl around it, all weighing a few hundred kilos. The reason that we do not find hammers like this is that people prefer hammers that they can use, so the design of the hammer has evolved over time to provide a good user-interface. This evolution is still continuing: within the last twenty years, metal handles with rubber grips have appeared to compete with wooden handles.

A computer is a strange type of tool in that the same box (or set of boxes) can be made to perform a wide variety of tasks. Although food processors also possess this property to some degree, they do not have the same versatility. At a distance [Figure 1.2(a)], the interface is seen as being the hardware. On a personal computer, this may consist of a screen on which text or pictures can be displayed, a keyboard and a mouse which the user can use to input data and commands. It also consists of any documentation used to describe and support the system.

Although the keyboard, mouse and screen are the most common ways of interacting with computers, there are other ways. In particular, sound, both in the form of voice input and of audible responses from the machine, is likely to become increasingly important.

Closer up [Figure 1.2(b)], the displays for different systems look different, require different inputs and do different things. One of the systems is used to maintain club membership records, one is used to draw diagrams, and the third is used to schedule vehicles from a vehicle pool. The user conducts a *dialogue* with the computer, in which their input produces some sort of response (e.g. the appearance of text or a change to a drawing) on the screen. The precise nature of the dialogue depends on the task to be performed.

The user-interface constitutes peoples' perception of the tool. The user does not need to know about nor is he interested in the metallurgy of the head of the hammer. If something about the crystal structure results in a hammer that is soft or bends, then it is a bad hammer. If a computer system does not do what it is expected to do, on the basis of what the user knows about it from the user-interface, then it is a bad computer system.

Tools are learned. Learning to use a tool involves two distinct processes:

- Learning *about* the tool, i.e. *what* tasks it can be used to do.
- Learning to *operate* it, i.e. *how* to perform those tasks.

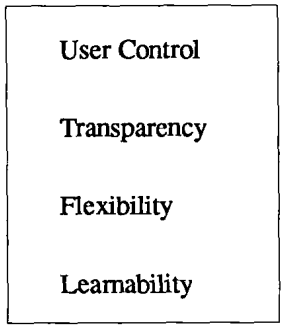



**Figure 1.3** Characteristics of a good tool.

This involves building up a *conceptual model* of the tool, which tells us what it is and how it behaves, and learning the actual skills to use it. In this way we learn that a hammer is for banging in nails and not for stirring soup, and we learn, largely through practice, how to hit a nail so that it goes in straight.

With a tool such as a hammer, the form of the tool is a significant help, since the form gives strong cues as to its function: there is a handle for holding, there is a flat area for hitting the nail with (so our aim does not have to be too accurate), and the claw at the back is for pulling out bent nails, so we pick it up and hold it the correct way without having to think "which way up does it go?"

With a computer system, the same thing should be true: the form of the presentation of the system to the user should tell them what it can do, and give clear indications of how.

An essential feature of a tool is that it *disappears*. When we are using it, we concentrate on the problem, and not on the use of the tool. When we are banging in a nail, we do not watch the hammer, we watch the nail. We have learned (after a few bruised fingers) how long the hammer is and how to manage the swing, and now do it instinctively. Similarly, if we are using a word processor, we want to concentrate on the document that we are writing, not on which key to press. We refer to this property as *transparency*.

Tools must also be *flexible*. Different people will want to use tools in different ways. Some of us are even left-handed. Thus a hammer can be used for banging in short nails as well as long nails, prising open cupboards, and even for stirring soup. A spreadsheet can be used to produce a balance sheet, predict stock-market trends, or for typing and printing a document if nothing better is available. If we can use a tool for unintended purposes (even if it does not do it as well as a different tool that we do not have) we regard that as a plus.

Unfortunately, many computer systems are not good tools. Dialogues are often clumsy and hard to learn, dictate to the user, and inhibit rather than assist the user's work by hiding information. The documentation describes in gruesome detail how to perform highly specific operations, but nowhere explains what the system can be used for nor relates the user's tasks to the keystrokes that are described. The result is frustration, irritation, and a tendency to use other, easier-to-use tools, such as pens and calculators.

The reason for this is historical (Figure 1.4). In the 1960s and 70s, computer systems performed well defined sets of operations, mainly routine transactions and massive scientific computations, and the interaction was handled by specially trained personnel

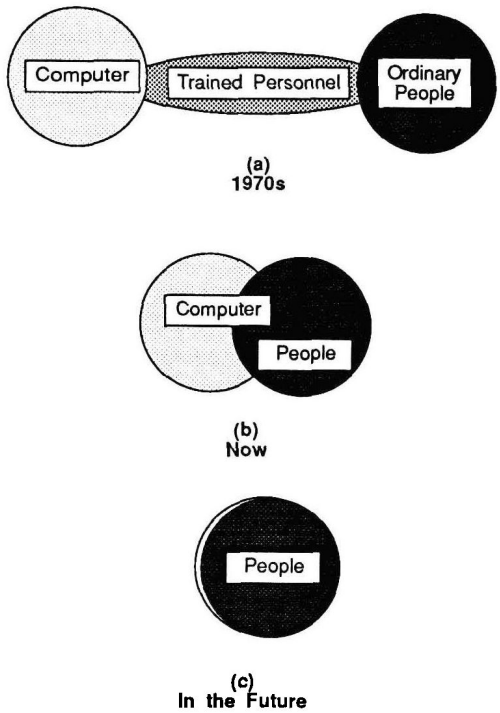



**Figure 1.4** People and computers.

for whom their use was the main activity in their job: programmers, operators, data-entry personnel, trained word processing operators, trained clerical staff, research scientists. Most people had no contact with computers, and even those that had did not use them for more than a few specific tasks; a research scientist might use a computer for complex mathematical modelling, but would still have their papers typed up on a manual typewriter. Since the machines were expensive, the major emphasis was on making efficient use of the machine, and not of the people's time, so that the user-interfaces were geared to whatever made the machine work most efficiently. Home computers had been thought of, but had failed miserably in the market place because they were too hard to use.

Thus, although computers were being used as tools, a better analogy would be with a crane or a bulldozer, and not with a hammer. The problem here is that what a crane driver regards as a good user-interface and what an amateur carpenter regards as a good user-interface are two different things. The crane driver has pride in being skilled at a difficult task and would probably resent any innovations that made it possible for everybody to drive cranes.

This is exactly what has happened with computing. The use of computers is now widespread. The number of people who do not use them in some way or another as part

of their work is becoming quite small, and they are using them in a different way. They are not using them because their use is the main activity in their job, but to support other activities, and a given individual will use them in a wide variety of ways, e.g. word processing, calculations, information retrieval, "what if" modelling, as well as routine transactions.

However, many of the user-interfaces are hangovers from past times. Worse, many of the ideas about what is a "satisfactory" user-interface are also hangovers, with many computer professionals seeing no need for change. One result is that the operating system currently supplied with the bulk of personal computers sold has an archaic command-driven interface that is hard to learn and stops many potential users at first base. That there is a demand for better interfaces is demonstrated by the success of the Apple Macintosh, a machine whose main selling-point is its user-interface.

The major impact of poor user-interfaces is economic. Computer hardware and software is cheap, but salaries are expensive. A marginal gain in efficiency through a better word processor, easier-to-use information retrieval, reporting or analysis software can

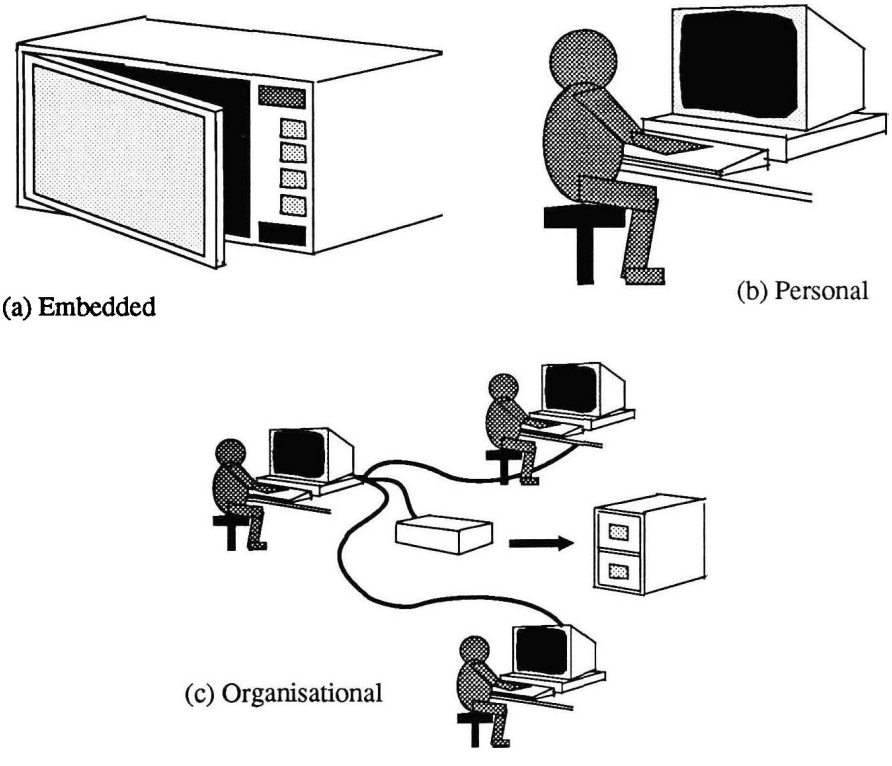



**Figure 1.5** Uses of computers.

rapidly mount up to thousands of dollars for an individual and hundreds of thousands or millions over a large organisation, for expenditures of a fraction of this amount.

For this reason, the design of better user-interfaces for computer systems has become an area of major interest. Research into user-interfaces covers a vast area, from cognitive psychology and ergonomics on one hand, to graphics and speech recognition. A major benefit already appearing from this research is the current generation of graphical user-interfaces, which started with the Xerox Star in the late 1970s, and which are currently incorporated into such products as the Apple Macintosh and the Microsoft Windows software. This book is concerned with the application of the ideas from this research and development work to the design of computer systems.

In the following sections we will expand on the idea of a computer as a tool by talking about what we do with computers, and about who uses them. We will then develop the idea of a conceptual model, and discuss in more detail the way in which the usability criteria can be applied to computer systems.

## 1.1 The Role of Computers

### 1.1.1 Uses of Computers

In order to design tools we must first consider what we use them for. There are three main uses for computers (Figure 1.5):

- As *embedded* systems in electrical devices such as microwave ovens and video recorders. These perform a small range of functions specific to the device into which they are built, e.g. in a microwave oven, the computer operates as a clock, a controller for the power level, can accept and remember commands affecting the cooking time and power level, and controls the alarm that is sounded at the end of the set cooking time.
- As *personal* systems, used for creating and editing documents, drawings, animation and music for personal use or subsequent distribution or publication, and for doing calculations.
- As part of *information* systems within an organisation or a society.

There can, of course, be overlap: an embedded system that operates traffic lights may feed data into a larger information system that monitors traffic flows, while one that is part of a synthesiser may be used to compose music; and documents created on a personal system may be sent to other people within an organisation, and so be part of that organisation's information system.

Within information systems, there are three main types of tasks:

- Transaction processing.
- Decision-making.
- Information retrieval.

A *transaction* is a routine task performed frequently and repetitively. It takes its name from financial transactions: the exchange of goods for money, as in a shop. Typical transactions handled by information systems include making a deposit at a bank, booking an air ticket, paying a bill.

Most large-scale computer systems are oriented towards transaction processing. The transactions are normally presented to the user as a well defined series of steps from which little or no deviation is permitted. Typical of this approach is the withdrawal of money from an automatic teller machine:

- Insert card.
- Type in Personal Identification (PIN) Number.
- Select transaction type (Withdrawal).
- Select account.
- Enter amount to be withdrawn.
- Remove card.
- Collect cash.
- Collect transaction slip.

These steps are always performed in order. If a mistake is made, it is usually possible to repeat the last step, or to cancel the transaction entirely and start again. If the machine detects an error it can force repetition of the step or terminate the transaction. It may also retain the card.

*Decision-making* requires a number of steps (Figure 1.6):

- The gathering of information on which to base the decision.
- The determination of possible actions.
- Modelling of the consequences of each of the courses of action.
- Choosing the action to take.

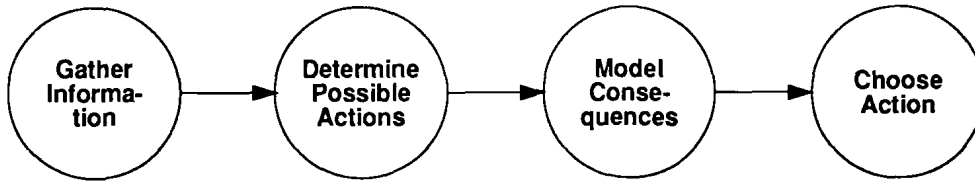



**Figure 1.6** Decision-making.

The role of a computer system in decision-making processes is as an aid. It is used to retrieve relevant data from one or more sources, to organise it in a manner relevant to the decision at hand, and to make quantitative assessments of the effects of some possible actions. Its use is normally relevant only to some aspects of the decision. The decision itself is made by the person. Computer systems performing this role are often referred to as *decision-support* systems.

Booking an airline ticket involves a decision: firstly, one must find out what flights are available; one then formulates a series of alternative actions (i.e. taking one of the available flights, going by train or bus, or staying home), weighs up these alternatives in the light of various criteria (e.g. whether the flight gets you there on time for your appointment, whether it means you have to get up too early in the morning, what the cost is for alternative routes), and selects the "best" alternative. The airline's booking system does not make this decision: it merely supplies you with the times of the available flights, and then records the result of your decision in the form of a booking on the chosen flight (if any). Thus the computer system is acting in a decision-support role. The actual booking is a transaction.

The final type of task is *information retrieval.* This is similar to the queries and reports that are the basis of the information-gathering process in decision-making, but we distinguish information retrieval processes from these in that the use of the information retrieved is of no concern to the system. Typical information retrieval systems are the bibliographic, financial, scientific and legal databases available for public access, and the systems installed in museums to give information on exhibits.

### 1.1.2 Types of User-Interface

The user-interfaces of different kinds of systems are adapted to their function. Embedded systems can have highly specialised user-interfaces, designed specifically for the task at hand, e.g. with a microwave oven, the user presses buttons or turns knobs to set cooking times and powers before starting the cooking process. The oven beeps when the preset cooking time is elapsed. In theory, these interfaces can be very good, because they do not contain any extraneous elements. In practice, many are very poor, because a few knobs or buttons are forced to perform a multitude of functions, and the user rarely learns all of them. Can you use all the functions on a microwave oven or an office telephone system? Do you even know what they are?

For personal systems, general-purpose hardware and software is used, e.g. a personal computer with keyboard, mouse and screen, running packages such as word processors, spreadsheets, and drawing packages. The user commonly has a great deal of flexibility in what they do and how they do it. With a drawing package, the mouse can be used to draw lines, rectangles and other shapes, to select line types and fill patterns, to move, reshape and resize them, simply by pointing to menu options or parts of the drawing displayed on the screen. A conventional keyboard is used to enter text for