

Alistair Sutcliffe

**Human–Computer
Interface Design**



Springer Science+Business Media, LLC

Human-Computer Interface Design

Human–Computer Interface Design

Alistair Sutcliffe

*Senior Lecturer
Centre for Business Systems Analysis
City University, London*



Springer Science+Business Media, LLC

Apple, the Apple Logo, Macintosh, MacDraw, MacWrite and MacPaint are trademarks of Apple Computers, Inc.

SAS is registered trademark of SAS Institute, Inc., Cary, NC, USA.

SQL is a registered trademark of IBM.

Wordstar is a registered trademark of the MicroPro International Corporation.

Prestel is a registered trademark of British Telecom.

Xerox is a registered trademark of Xerox Corp.

© Alistair Sutcliffe 1989

Originally published by Springer-Verlag New York in 1989.

All rights reserved. No part of this publication may be reproduced or transmitted, in any form or by any means, without permission.

First published 1989

MACMILLAN EDUCATION LTD

London and Basingstoke

Sole distributors in the USA and its dependencies

Library of Congress Cataloguing-in-Publication Data

Sutcliffe, Alistair, 1951–
Human-computer interface design.

Bibliography: p.

Includes index.

1. Computer software–Human factors. 2. Computer input-output equipment. I. Title.

QA76.76.H85S88 1988 004'.01'9 88–29456

ISBN 978-1-4899-6751-0 ISBN 978-1-4899-6749-7 (eBook)

DOI 10.1007/978-1-4899-6749-7

Contents

<i>Preface</i>	viii
<i>Acknowledgements</i>	xi
1 Introduction	1
1.1 What is Human–Computer Interface Design?	1
1.2 Why Design Interfaces?	2
1.3 Human–Computer Interface Design and Computer Science	4
2 User Psychology	6
2.1 Understanding Users	6
2.2 Vision	11
2.3 Hearing	19
2.4 Learning and Memory	24
2.5 Thinking and Problem Solving	34
2.6 Control of Human Information Processing	42
2.7 Principles of Human–Computer Interaction	45
2.8 Summary	47
Further Reading	48
3 Interface Analysis and Specification	49
3.1 Task Analysis	49
3.2 Analysing User Characteristics	53
3.3 User Models and Views	56
3.4 Task and Job Design	59
3.5 System Environment and Support	64
3.6 Interface Design Styles	66
3.7 Review of the Type of Interaction	75
3.8 Selecting the Interface Design Style	76
3.9 Summary	79
Further Reading	80

4	Theoretical Approaches	81
4.1	Command Language Grammar	81
4.2	Other Grammatical Specifications	88
4.3	Diagrammatic Specifications	89
4.4	Cognitive Complexity Theory	90
4.5	Comparison of Specification Methods	94
4.6	Summary	96
	Further Reading	96
5	Dialogue Design	97
5.1	Designing the Interface Structure	97
5.2	Principles of Good Design	101
5.3	Putting Principles into Practice	103
5.4	Checking the Design	106
5.5	Summary	106
	Further Reading	107
6	Presentation Design	108
6.1	Screen Design Procedure	108
6.2	Detailed Display Design	111
6.3	Summary	117
	Further Reading	118
7	Data Entry Interfaces	119
7.1	Data Entry Guidelines	119
7.2	Forms Design	121
7.3	Form-filling Interfaces	127
7.4	Alternative Data Entry Techniques	133
7.5	Summary	136
	Further Reading	136
8	Data-Display and Data-Retrieval Interfaces	137
8.1	Data-display Guidelines	137
8.2	Character Data Displays	139
8.3	Data-query/Data-retrieval Displays	140
8.4	Graphical Displays	144
8.5	Reports	148
8.6	Summary	155
	Further Reading	155

9	Computer Control Interfaces	156
9.1	Control Dialogue Guidelines	156
9.2	Simple Control Dialogues	157
9.3	Menu Interfaces	158
9.4	Function Keys	162
9.5	Icons	164
9.6	Direct Manipulation	166
9.7	Windows	168
9.8	Command Languages	169
9.9	Natural Language	174
9.10	Summary	180
	Further Reading	180
10	Development of Human–Computer Interfaces	181
10.1	User-centred Design	181
10.2	Evaluation of Human–Computer Interfaces	182
10.3	Adaptive and Intelligent Interfaces	186
10.4	User Interface Managers	187
10.5	Formal Specification of Dialogues	190
10.6	Summary	191
	Further Reading	192
10.7	Postscript	192
	<i>References and Further Reading</i>	195
	<i>Index</i>	202

Preface

The motivation for this book started when I introduced a course on Human–Computer Interaction in the BSc Computation degree at UMIST and began to look for a course textbook. At the time (1984) there were few books on the subject as a whole and no really suitable textbook for undergraduate courses. Although that situation is now changing, I find my motivation undiluted for another reason. Increasingly the human–computer interface has become part of software development which practitioners in industry and academia recognise as important, yet the subject is taught in very few computer science courses. Unless we educate the system developers of tomorrow—that is, the systems analysts and programmers who are being trained now—about human–computer interaction, there is little chance of changing the current practice of poor interface design. Accordingly my chief aim is to bring the message of human–computer interaction to computer science students.

This perspective is worth some comment because the whole field of human–computer interaction is relatively young and a consensus about what should be taught within the subject area has only recently become clearer. I shall therefore explain my motives in writing this book in more depth.

The primary aim is to give computer scientists knowledge of the issues in human–computer interaction, and help develop the skills needed to design better human–computer interfaces. As computer scientists are the major creators of software and hence human–computer interfaces, it is vital that they acquire knowledge and good practices of interface design. If this part of their education is neglected poor interfaces will continue to be foisted on users, making systems frustrating or unbearable, even though the internal software might be a perfect example of good software engineering practice.

I have attempted to place interface design into a framework of software development by drawing on methods from systems analysis and design as well as ideas in human–computer interaction. Interface design is ultimately part of a wider design process for the whole system and should be integrated with mainstream systems development. Accordingly I place interface design and its components within the systems design life cycle.

The objective of teaching interface design begs the question of what to teach. Interface design is about designing human–computer interfaces for people. It seems common sense for designers to be knowledgeable about the subject of their designs, in this case people. The starting point therefore was to provide some appreciation of human psychology which is of importance to human–computer interaction: principally, perception and cognition which cover how we see, hear, think, learn and remember. It was not my intention to turn computer scientists into psychologists, hence the treatment of psychological material has had to be brief and is presented without extensive reference to background research.

Psychologists, being empirical scientists, are quite correctly guarded in their assertions and conclusions. Computer scientists on the other hand deal in a more finite world and are unaccustomed to unsure knowledge and guarded assertions. This difference in view has caused some conflict in the field of human–computer interaction, with computer scientists criticising psychologists for not offering firm opinions; while psychologists criticise the computer scientists' thirst for simplistic views in a subject which is extremely complicated. To please both views is a task somewhat akin to playing hop-scotch on a mine field. At the risk of offending my psychological colleagues, in this book I have taken the computer scientist's viewpoint. In doing so I have had to gloss over the controversies which surround some topics in cognition and perception.

In addition to providing psychological background to the subject, this book aims to teach a methodical approach and practical skills in interface design. The material is organised into four sections. Chapters 1 to 5 cover the psychological background, establishing general principles of human–computer interaction and describing a method of interface design. This is followed by chapters 6 to 9 which give practical design advice for data entry, data display, and command and control interfaces. Chapter 10 concludes with an examination of the place of interface design within systems analysis and design, and a brief survey of current research topics in the subject.

Human–computer interaction is a large field of endeavour with ill defined edges. In an undergraduate text it is impossible to cover the whole field; I have therefore been selective in the topics for study. Many issues which are more hardware in nature are not treated in depth; also system environment issues, social consequences of computer systems, and experimental practices for interface evaluation receive little space. These topics are more than capably investigated by others whose works are cited in the references.

It is a pleasure to acknowledge the help of Bill Black, Graham Hitch, William Edmonson and Ken Eason who have commented on the contents and various parts of the manuscript. Any inaccuracies which remain are of my own making. Finally, my last motivation for taking up the author's pen

my own making. I am also indebted to Gillian Martin for her efficient proof reading and her tolerance and support during the creation of this book. Finally, my last motivation for taking up the author's pen was self-interest. This book was prepared on a variety of word processing software with inadequate interfaces. If future authors have better tools, I will have succeeded in my quest to stamp out user-vicious software.

A G. Sutcliffe

December 1987

Acknowledgements

The author and publishers wish to thank the following who have kindly given permission for the use of copyright material.

Academic Press, Inc. for an illustration of a speech spectrogram by Peter Bailey from *Fundamentals of Human Computer Interaction* ed. A. Monk, 1985, Fig. 12.3. [Figure 2.10]

Academic Press (UK) Ltd for illustrations from 'An Approach to the Formal Analysis of User Complexity' by D. Kieras and P. G. Polson, *International Journal of Man Machine Studies*, **22**, 1985, pp. 365–394, Tables 2, 3 and 4, and Figs 5 and 6; 'The Command Language Grammar: A Representation for the User Interface of Interactive Computer Systems' by T. P. Moran, *International Journal of Man Machine Studies*, **15**, 1981, pp. 3–50, Figs 5, 6, 7, 11, 12, 16 and 17. [Figures 4.1, 4.2, 4.3]

Terry Allen Designs for illustration of the Hermann grid. [Figure 2.5]

Apple Computer, Inc. for illustrations of MacDraw™, MacWrite™, MacPaint™ and Hypercard™ screens. [Figures 3.7, 3.10, 8.5b, 9.4, 9.7]

Association for Computing Machinery for illustration from 'Put-That-There: Voice and Gesture at the Graphics Interface' by Richard A. Bolt, *Computer Graphics*, **14** (3). Copyright © 1980 by Association for Computing Machinery, Inc. [Figure 8.1, top].

G. A. Fisher for illustrations of a duck/rabbit series from 'Materials for Experimental Studies of Ambiguous and Embedded Figures', *Research Bulletin of the Department of Psychology*, University of Newcastle Upon Tyne, No. 4. [Figure 2.7, part]

Glydendal for illustration of vase/faces from *Synsoplevede Figurer* by Edgar Rubin. [Figure 2.7, part]

MicroPro International for illustration of the WordStar^R and WordStar^R 2000 products. [Figure 9.3]

Oxford University Press for illustration of hawk/goose from *The Study of Instinct*, by N. Tinbergen, The Clarendon Press, 1951 and part of illustration of mach bands from *Seeing: Illusion, Brain and Mind* by John P. Frisby, 1979, Fig. 158. [Figures 2.5, 2.7 part]

SAS^R Software Ltd for an illustration of SAS/GRAPH software. [Figure 8.5a]

Van Nostrand Reinhold, Inc. for an illustration of the photograph of a Dalmation by R. C. James. [Figure 2.8]

Every effort has been made to trace all the copyright holders but if any have been inadvertently overlooked the publishers will be pleased to make the necessary arrangement at the first opportunity.

1 Introduction

This chapter sets the scene for interface design by first placing interface design within the context of human factors and human–computer interaction and then exploring some justifications for why it is necessary to spend time and money designing human–computer interfaces.

1.1 What is Human–Computer Interface Design?

New areas of endeavour in any discipline have ill defined boundaries which take some time to become stable. During this process the important issues in a subject become clear and the span of topics which properly constitute the subject becomes defined. Unfortunately, interface design, because it is relatively new, has ill defined boundaries, a variety of names and a great number of topics which may be considered. Therefore, I shall begin by drawing some boundaries around the topics covered in this book and look at the wider perspective of the subject as well.

Generally, human–computer interface design falls into the subject area called Human–Computer Interaction or the Man–Machine Interface. This spans the two older disciplines of computer science and psychology but also draws on material from linguistics, ergonomics and sociology. Human–computer interface design, in the sense of this book, is the process of designing interface software so that computer systems are efficient, pleasant, easy to use and do what people want them to. The human–computer interface is more than just the software and concerns hardware, the system environment and human organisation, but because this book is aimed primarily at computer scientists, software is the prime focus. Although this book concentrates on interface design from the computer scientists’ point of view, and within that constraint focuses on the design of the software part of human–computer interfaces, it is important to realise that this is only part of human–computer interaction.

One particular discipline, ergonomics, has made a considerable contribution to interface design in both the broad and narrow perspectives over many years. Ergonomics, which is called Human Factors in the USA, is a branch of applied psychology which aims to improve the design of machines for people. In doing so, it is intimately involved with understand-

ing the process of human-computer interaction. While this book will draw on some material from ergonomics, constraints of space mean that many ergonomically oriented interface issues (such as workplace design and hardware ergonomics) cannot be covered.

The importance of this area of research has been recognised in the British Government's information technology research programme, the Alvey initiative, which placed the Man-Machine Interface on equal terms with three other branches of computer science: Software Engineering, Intelligent Knowledge Based Systems, and Very Large Scale Integration. As a result of the Alvey programme, a large amount of research is currently under way into interface design problems. Emphasis has also been placed on the subject in the research programmes of the EEC (ESPRIT), the USA and Japan.

Human-computer interaction research covers a broad field from interface hardware, the environment in which the interface is situated, and the effect of the interface on people, both individuals and groups, to the software issues of building interface software and tools to help construct interface software itself. A broad classification of the field subdivides into background issues, methodological issues, design practices and tool construction, giving the following topics:

- Understanding the essential properties of people which affect their interaction with computers
- Analysing what people do with computer systems and their interfaces; understanding the user's task and requirements
- Methods of specifying how the interface should function, how it should respond to the user, and how it should appear
- Design of computer interfaces so they fit the properties of people and their objectives
- Design of tools to help designers build better interfaces
- Evaluating the properties of human-computer interfaces and the effect of systems on people

These topics are naturally inter-related, so the theoretical background of the subject, based on psychology, should have an impact on the methods practised and on the tool environments constructed to help interface developers. Likewise the process of analysis, specification and design are necessarily interlinked.

1.2 Why Design Interfaces?

Before investing time, money and effort in any endeavour, the prudent will always ask if it is worth the investment. Interface design will undoubtedly

add to the cost and effort of developing computer systems, so it is worth devoting a little space to the question of justification.

Interface design has been present in the computer science literature and industry for a decade or more; see, for instance, the early text by Martin (1973). People have realised and complained for a long time that computer systems are difficult to use, obtuse and jargon-ridden. By and large, users had to put up with this state of affairs because computer programmers took no notice of their complaints. The rise of human-computer interaction as an active discipline correlates well with the rise of the microcomputer. A plausible explanation for this is that for the first time computers and their software became mass circulation commodities for ordinary people. People rejected the jargon-ridden, unreliable offerings of earlier systems because they had a choice in an open market place. Early interfaces to microcomputer software are user-vicious by today's standards, but compared with their contemporary mainframe rivals they were way ahead, a state of affairs which still generally applies at the present time.

Interface design became important because pleasant, attractive, easy-to-use software sells well. But interface design is important whether a system is to be sold or not. The interface is the part of the system which the user sees, hears and communicates with. Depending on his or her experience with the interface, a computer system may succeed or fail. It is irrelevant how well engineered the software code is and how sophisticated the hardware is; a bad interface can ruin an otherwise excellent system. On the other hand, a good interface can save poor software and make a system acceptable.

Computing systems are becoming increasingly interactive. As they do so, the amount of code which is written for input and output (that is, the interface) has risen. It is estimated that most commercial decision support and information systems have between 70 and 80 per cent of their code devoted to interface handling. In on-line systems it is the interface which is not only the critical part but also physically the largest part. Good design is vital.

The cost justification of interface design is not hard to argue, although statistics are hard to find (as is the case with software reliability). Poor interface design can have the following consequences:

- *Increased mistakes in data entry and system operation.* Mistakes cost money to rectify and errors which go uncorrected can have very damaging consequences if decisions are taken on the basis of incorrect data
- *User frustration.* This may be manifest in low productivity, employee stress, sabotage of the system or simple under-utilisation of the system. All these consequences cost money
- *Poor system performance.* The system may not handle the volume of throughput it was designed for, or the accuracy of output may not agree

with the specification. Poor interface design makes it too cumbersome to use or too obscure to learn. Extra resources and money have to be put into the system

- *System failure because of user rejection.* This may seem to be an extreme case considering the tolerance of users to appalling software, but it happens. The US Dept of Defense ascribes its worst system failures to a combination of poor interface design and inadequate system requirements analysis

Good interface design is essential for good system performance. All the above problems, inherent in poor designs, cost money either to fix or in terms of operating costs. In addition to system performance considerations, there is the question of user tolerance to poor interface design. In the past, users have tolerated much poor interface software. This is unlikely to be so in the future. Many people are becoming exposed to microcomputers with attractive, pleasant-to-use software. Such software will become a norm which cannot be ignored by developers of in-house mainframe systems. Defending poor interface design is becoming harder to justify. Both the program and systems design communities have recognised the vital importance of good design, and the application of design to human-computer interfaces is long overdue.

Design is not an intuitive process. True, some designers have a flair for finding innovative and good designs, but most people do not. Design is a process which has to be taught. It is a matter of applying knowledge to a design problem. The knowledge may be guidelines and principles bound up in a method which shows designers how to proceed; then as experience and design practice mature, more formal procedures and specifications may be introduced. In the case of interface design, the knowledge is in the form of guidelines and principles derived from psychology, the science of understanding the customers, that is, the people who are computer users.

1.3 Human-Computer Interface Design and Computer Science

The Human-Computer Interface (HCI) permeates many parts of computer science and should be part of any system development which involves people as users. However, within computer science the most closely related areas are systems analysis and design, and artificial intelligence. The methodological part of HCI is concerned with many issues familiar to systems analysts. What HCI workers call task analysis, systems analysts call requirements and current logical system analysis. The approach, methods and emphasis may be different but both disciplines are trying to establish and specify what the users want the computer system to do for them. Both system developers and HCI designers build software, the difference being one of perspective; the HCI designer concentrates on the

user whereas the system developer is more concerned with data and functional analysis. In the future this distinction should disappear.

Artificial intelligence shares the HCI interest in human cognition. Both disciplines construct models of reasoning and problem solving, although again the approach and perspective may be different. In HCI design, understanding human reasoning and memory is important so that systems can be built to accord more closely with natural human properties and to adapt to human individual differences. The artificial intelligence perspective is more motivated towards building machines with human-like thinking, memory and learning abilities in the long term, and finding efficient problem-solving mechanisms in the short term. Both artificial intelligence and HCI may come to use the emerging discipline of Cognitive Science as their theoretical inspiration as this subject embraces computational approaches for the study of human processes.

Human-computer interaction bridges, to an extent, the gap between systems analysis and design, and knowledge-based systems, having a methodological similarity with the former and a theoretical basis largely shared with the latter. In addition, HCI specification shares commonalities with software engineering. Interfaces have to be specified so that their behaviour can be predicted and described in an exact manner; to do so requires precise methods of specification, many of which have been borrowed from software engineering. As the human-computer interface will comprise a significant amount of the overall software in a system, it is natural that computer scientists should wish to apply rigorous standards to it, as they do to non-interactive software. These issues are returned to in chapter 10.

Design of the human-computer interface is a necessary activity in nearly every system which is designed. The subject fits with more established disciplines within computer science and should, as it matures, become increasingly integrated within the kernel of computer science subjects.

2 *User Psychology*

This chapter gives an overview of cognitive psychology which is relevant to human–computer interaction. It starts with how we perceive information from the environment with the senses of sight and hearing and then progresses to understanding the information we receive. Memory is then investigated: how information is coded and possibly stored, with the limitations of human memory. This leads on to mental activity and how we reason and solve problems, and the control of mental activity as attention is reviewed together with more general topics of stress and fatigue. The chapter concludes with a summary of the principles of interaction based on knowledge of human psychology.

2.1 **Understanding Users**

Throughout this chapter a metaphor of a human computer will be used, with the objective, I hope, of making the workings of the human brain easier to understand. Please note that this view is just a model of how things may operate in the human mind based on psychological study; it is by no means a definitive statement of how the human brain is structured or how it operates. Such topics are still active areas of psychological research. Viewing the brain in terms of processors, memories and messages is a convenient analogy, nothing more.

Basic anatomy of the human processor

The human brain is composed of a vast number of nerve cells, estimates varying around 15 billion. Nerve cells are the basic elements of human processing and memory. Each cell is a single small living unit (see figure 2.1) bounded by an envelope which keeps it all together called the *cell membrane*. The important quality of nerve cells is that they are capable of electrical activity. The electrical activity is not the same as in electrical circuits; instead it is electrochemical activity caused by different concentrations of metallic ions separated by the cell membrane. The electrical activity is caused when ions are allowed to flow across the membrane to

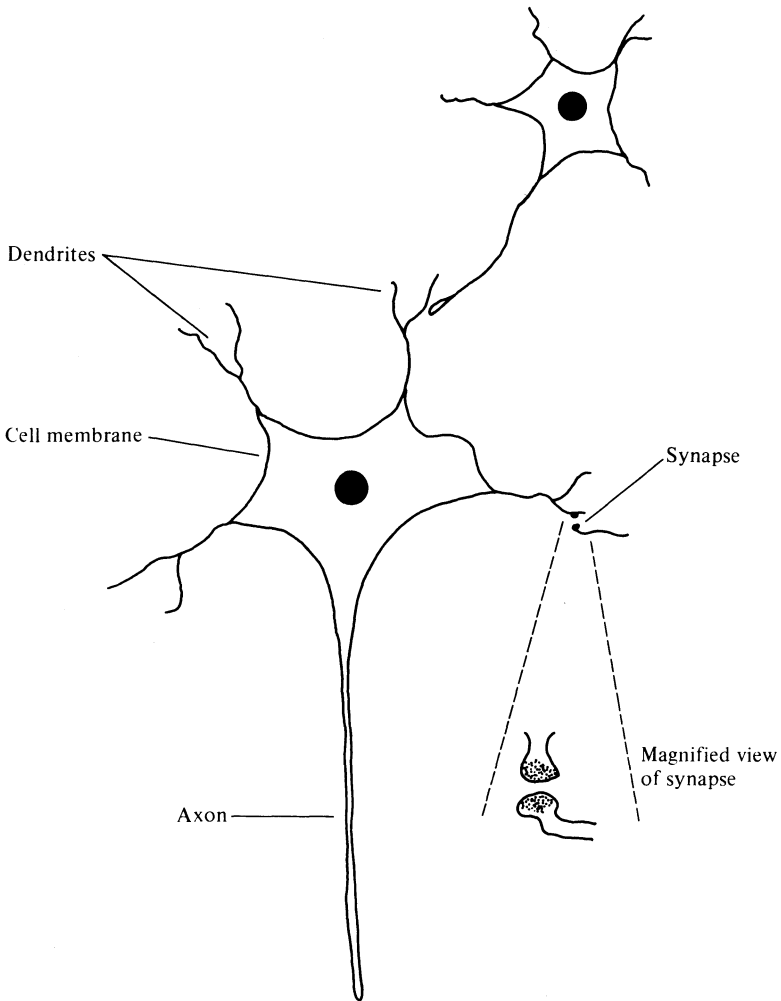


Figure 2.1 Microscopic view of a single nerve cell showing inter-cell connections. Other nerve cells send impulses which either excite or inhibit the receiving cell, making it more, or less, likely to fire.

re-adjust differences in concentrations. This causes a change in the electrical potential across the membrane, called *depolarisation*.

Depolarisation signals a state change in the nerve cell, creating the digital 1/0 states necessary for computation. Unlike transistors, nerve cells do not retain a state change indefinitely. As soon as depolarisation has occurred, the nerve cell tries to return to its previous resting state by pumping metallic ions across the membrane to re-establish the concentra-

tion difference. Once the concentrations have been restored the nerve cell can fire again; however, re-adjustment of concentrations takes a short period, so continuous activity is not possible.

Nerve signals (called *impulses*) travel along nerve cells, but to transfer to the next cell they must cross an inter-cell gap. This gap is very small (2–5 microns) but so is the electrical voltage, which means it cannot jump the gap. Inter-cell transmission is by chemical means. When an impulse reaches the nerve end it triggers the release of a chemical which has to diffuse across the gap. The chemical then stimulates electrical activity in the next cell, causing it to depolarise. This electrochemical activity means that the speed of nerve messages is slow compared with the speeds of electrical signals in computers.

Electrical signals from nerve cells usually come as a series of blips, each firing being a transient 010 state change as depicted in figure 2.2. The

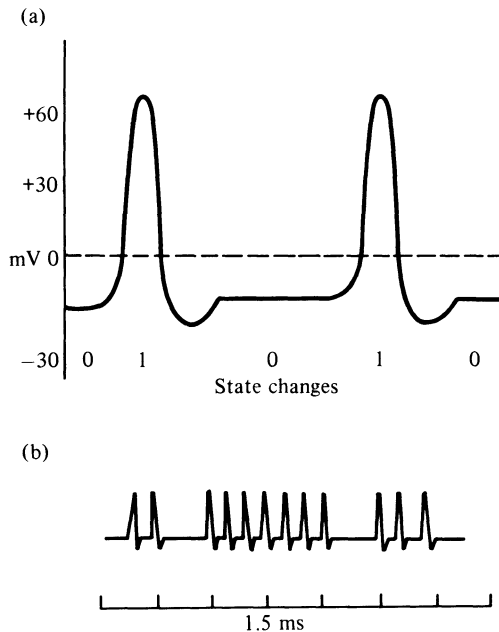


Figure 2.2 View of a nerve cell impulse recorded as a cell fires. The electrical potential changes from resting level of -10 mV by about 60 mV, then overshoots to -5 mV before returning to the resting level. The whole event, called a nerve impulse, forms the basis of a 010 state change. (a) Electrical changes in a nerve cell. (b) Typical recording from a nerve cell showing a series of impulses. Frequency of nerve impulses can code analogue signals. The second recording shows a series of nerve impulses within a 1.5 ms time period.

coding of messages between nerve cells is different from that of electronic circuits because nerve impulses are transient and on account of the effect they have on receiving cells, either increasing or decreasing the tendency of a receiving nerve to fire. Inter-nerve cell connections can therefore be *inhibitory* and damp down a cell's activity, or *excitatory* and stimulate it. Whether a nerve cell fires or not and its rate of firing depend on its innate tendency to fire and the inhibitory and excitatory influences upon it. As most nerve cells in the brain have an average of 150 connections, this composite influence can be very complex and is capable of expressing analogue effects as well as being very finely tuned. The complex signalling permits much more complex coding than in digital electrical signals. As a result a branch of computer science, *neural computing*, has developed to build artificial versions of nervous processors. The essence of nerve cell based communication is:

- An electrical change of the order of 30 mV can be detected in nerve cells when they fire. This is called a *nerve impulse*. It lasts for a few milliseconds
- The nerve cell restores itself to its original state a short time after firing
- The change can be repeated many times a second
- A single nerve cell signal is a transient 010 state change
- Nerve cell signals are a mixture of digital and analogue. Unlike computer logic circuits the digital signal is very transient and analogue signals can be generated by differences in frequency, that is, many or few impulses per second
- Nerve cells can signal a change of state easily but continuous activity is not possible; this creates problems for signalling steady-state conditions
- Nerve cell signals are slow in comparison with electrical circuits (milliseconds compared with nanoseconds) because the signal is an electrochemical change

The effect nerve cells have on each other is determined not only by the quantity of impulses they receive from other nerve cells but also by the sensitivity of their reaction. Some cells require a lot of impulses to excite them sufficiently to fire, while others are tuned to hair-trigger levels. In this way nerve cells can imitate AND and OR logic gates and other familiar computer logic components, as illustrated in figure 2.3, as well as coding analogue signals.

Nerve cells in the brain are highly interconnected in a very complex network. Even if psychologists have not been able to unravel the wiring diagram of the brain, we do know something of its higher-level components. Processing is divided into right and left halves. The right-hand side of the brain is generally considered to be responsible for the more creative and artistic functions while the left-hand side has the more logical reasoning faculties. There are also discrete areas for sight, hearing, touch and the other senses, memory and areas devoted to coordinating muscles;

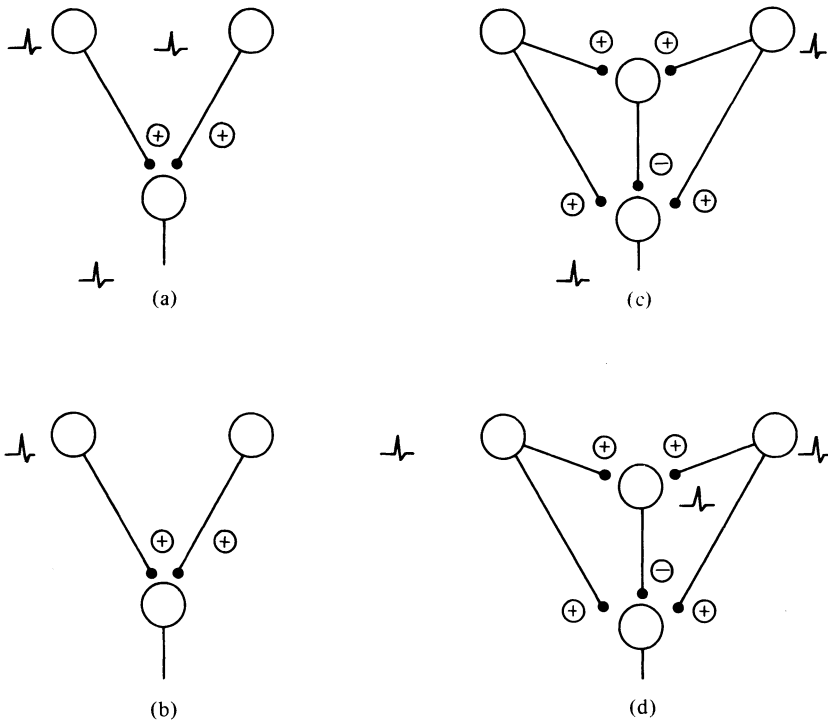


Figure 2.3 Connections between nerve cells to form a human logic circuit. The effect on the receiving cell depends on both the frequency of the impulses received from sending cells and the type of connection which may be excitatory or inhibitory. (a) and (b) show an AND gate, (c) and (d) an OR, respectively in firing and non-firing conditions.

figure 2.4 illustrates the anatomical geography of these areas. Beyond a top level functional division, little definite information can be given about the microstructure of the human brain. The human wiring diagram is infinitely more complex and subtle than the most advanced microprocessor and is still poorly understood. Attempts to follow the real architecture of the brain at lower levels are at present unrewarding; therefore to further our understanding of the human machine we shall use an abstract model, that is, an interpretation of how the logical processing units in the brain may work.

Cognitive models

These models have been devised by psychologists to explain human mental activity using an analogy of computer processing. It is important to remember that models are only an abstraction; the final story of how the

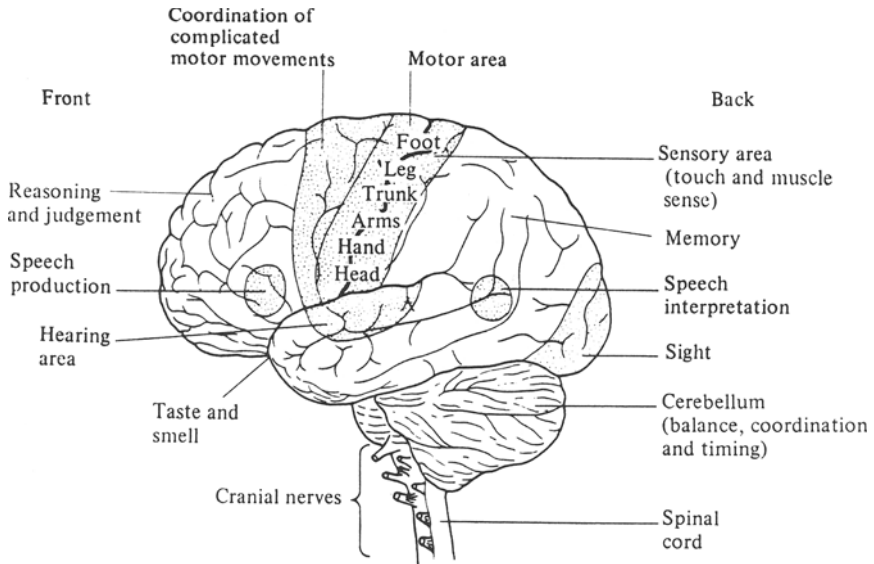


Figure 2.4 Functional anatomy of the brain.

human machine works will be much more complex. Cognitive models, however, are useful because they illustrate the advantages and limitations of the human machine, qualities which can be inferred from experimental evidence. In interface design we will need to take these qualities into account.

In the following sections, perception and cognition will be explored using an information processing model based on the work at Xerox by Card *et al.* (1983). *Perception* is the process of receiving information from the outside world, while *cognition* is the mental activity we describe in everyday terms as reasoning, problem solving, thinking and learning. The boundary between the two is blurred because as we receive information, we also interpret it, and use it to problem-solve. We shall look at the receptive processes first.

2.2 Vision

Vision is the dominant sense we use when interacting with computers, which has implications for VDU screen design and other display devices.

Perception poses three problems for the human machine:

- Receiving an external stimulus, in this case the electromagnetic radiation of light
- Translating the stimulus into nerve impulses in a manner faithful to the stimulus
- Attaching meaning to the stimulus

To resolve the first problem, nerve cells have to be made sensitive to light. Light is a form of electromagnetic radiation with a wavelength between 400 and 700 nanometres (nm). Other forms of radiation have longer wavelengths, for example, infra-red radiation or heat, 1000 nm, or shorter wavelengths such as X-rays. Within visible light colour subdivisions are defined by wavelength; at the longer wavelengths (650–700 nm) is red light, progressing through the colours of the spectrum to blue light at short wavelengths (300 nm).

The other physical property of light is its intensity, a measure of how much energy it contains. Unfortunately, human perception of light rarely bears a close relationship to the actual physical properties, the disparity being a testament to the pre-processing of physical light by the eye. Consequently, brightness of light is not just its physical intensity but is also conditioned by the difference between light intensities in an image and what we have seen previously. The subjective judgement of brightness also overlaps with measures of colour, as we see colours mixed with white as brighter than darker ones.

Light has two objective measures, luminance and contrast, and one subjective measure, brightness. *Luminance* is a measure of the light reflected from a surface. This is a composite of the amount of light falling upon a surface and the quantity reflected from the surface; in general, dark surfaces absorb more light, light ones absorb less light. Luminance as measured by photographic lightmeters is expressed in candelas per square metre (cd/m^2). *Contrast* measures the difference in luminance between two surfaces and is expressed as the ratios

$$\text{Contrast} = \frac{L_{\max} - L_{\min}}{L_{\max} + L_{\min}} \text{ or } \frac{\{L(\text{object}) - L(\text{background})\}}{L(\text{background})}$$

The $L_{\max/\min}$ formula ratio gives a measure between 0 and 1 for low to high contrast. Hence to make an object stand out in an image, a high overall luminance is desirable (L_{\max}) and a large difference between the object and background. This gives our intuitive feeling of high contrast in bright sunlight.

Brightness, on the other hand, is a subjective measure; although it may have a relationship to luminance this is not always reliable. Contrast can play tricks with our judgement, with the result that figures of identical luminance can be discriminated as having different brightnesses, as illustrated in the contrast bands and intersection contrast illusions in figure 2.5.

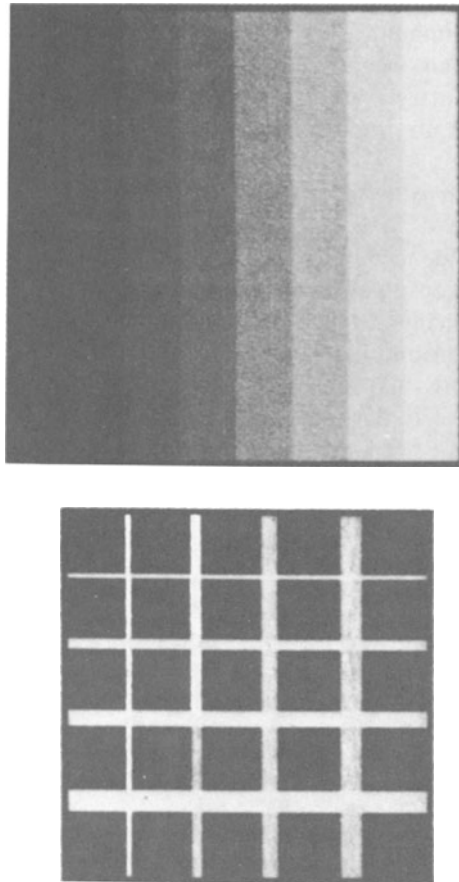


Figure 2.5 Perceptual illusions in contrast: Mach bands and the Hermann grid. The bands do not have the sharp change in brightness which we see and the shadows in the grid intersections do not really exist.

Brightness is measured by discrimination tests on thresholds or just noticeable differences. The limit of discrimination for human vision can be summarised by a ratio:

$$\frac{dL}{L} = k$$

where dL = threshold luminance

L = background luminance

k = a constant, roughly 0.01 to 0.02 for VDU displays.

This gives a foreground/background ratio between 1:100 and 1:50, hence as the background luminance is increased, objects become increasingly difficult to see. Our visual acuity, however, is not just dependent on luminance and contrast; other factors such as background lighting and image composition are important.

Visual acuity and sensitivity

Visual acuity is influenced by several factors. There is the complexity of the image itself, the intensity of the light, and image colour. Low light intensity makes images difficult to resolve. If the object is illuminated on a VDU screen, high background light intensity also makes resolution worse.

Absolute human visual sensitivity is remarkable, as the human eye can see in almost complete darkness, although the threshold of vision, that is, the smallest quantity of light that can be seen, increases with age. Even though people can see light at low intensities, they can resolve little detail and for normal working good illumination is required. This has implications for VDU displays. The advantages of good luminance in VDU displays are:

- Acuity increases with better luminance
- Better luminance means a smaller aperture in the eye which increases the depth of field. In the eye, aperture is controlled by the iris; the effect is the same as reducing the camera stop from F5.6 to F8, which gives a better depth of focus
- Better luminance means reflected light is less noticeable and hence less distracting

On the minus side, increased luminance makes VDU flicker more obvious and direct glare may become uncomfortable. Visual flicker is caused by the eye discriminating changes in an image over a short time period. If the change happens quickly enough the eye assumes a continuous state and does not differentiate between each image; this quality, called the flicker fusion frequency, happens at approximately 32 images/second, and the continuous-state illusion is exploited in motion picture photography. At slower rates of change the eye starts to notice the difference, which on a VDU screen becomes an annoying flicker. VDU flickers depend on the refresh rate, that is, the number of times a second the screen is scanned and the image redrawn. Usually VDU monitors use rates around 50 Hz (scans per second) which avoids flicker in most circumstances except for high luminance displays.

Human visual acuity is quite remarkable but individually very variable. Most people can resolve gaps of 2 mm at a distance of 2 metres but this tells us little about how people see meaningful shapes. Of more importance for interface design is resolution of more complex shapes and letters. The optician's test measures optimal visual ability as resolving letters 20 mm

high on the bottom row at 6 metres even though average ability is only capable of resolving 40 mm letters. Few people have perfect vision, so display design should accommodate average human abilities. One design implication of acuity is the size of text characters.

The size of printed letters is measured in points, a point being roughly $1/72$ of an inch; thus 10 point type has letters with an approximate height of $10/72$ or 0.14 of an inch. Printed text usually ranges between 18 and 8 point; anything smaller than 8 point is difficult to read for a long period of time and letters smaller than 6 point are almost impossible to resolve for reading purposes.

Colour sensitivity varies between individuals and between colours. Most people can see yellows better than reds, greens and blues; however, colour blindness should also be considered. Approximately 9 per cent of the male population have some colour blindness, and the inability to discriminate reds and greens is most common. Discrimination between colours is best in the mid range of the spectrum where the discriminable difference for shades of colour in terms of wavelength is 1 nm; towards the edges of the spectrum this rises to 20 nm. However, apart from simple images, discrimination also involves recognition. Behind the statistics of human vision lies a complex apparatus of reception and image interpretation, which we shall now consider.

Visual processing

Our eyes are sensitive to light because of photosensitive pigments, chemicals which change in response to light and create an electrical signal for transmission by the optic nerves. The chemicals, rhodopsin for colour, and iodopsin for black and white, are present in cells in the retina or back of the eye. The receptive cells, rods for black and white and cones for colour, have an irregular distribution in the retina. Rods are more concentrated around the periphery while cones are concentrated at the centre with the maximum cell density in the fovea which is the natural point of focus on the retina.

Light enters the eye through the lens and is focused, upside down, on the retina as illustrated in figure 2.6 which depicts the anatomy of the eye. The retinal cells react to the patterns of light in the image, firing with a frequency corresponding to the light intensity. The nerve cells are organised in a mosaic of small groups to cover the whole image. So far the mechanism may appear to be similar to a photographic process but in fact vision is very different.

Before images are transmitted to the brain, the eye does a considerable amount of image enhancement. The human visual system is much better at dealing with variation in light intensity than even the most sophisticated

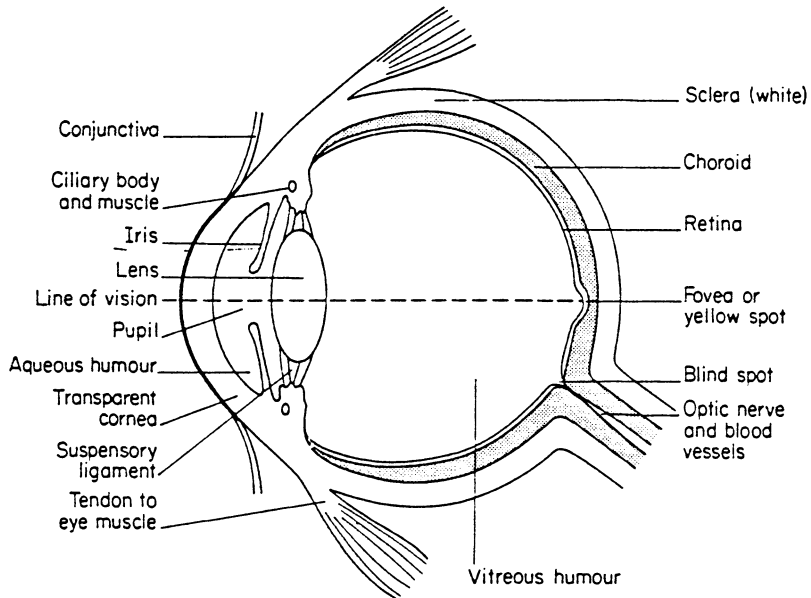


Figure 2.6 Anatomy of the eye. Images are focused by the lens on to the photoreceptive retina at the back of the eye.

cameras. This is because the eye has an automatic intensity adjustment device which turns the nerve cell sensitivity up in dark conditions and down in bright light. Another example of pre-processing is in the treatment of boundaries. The retina has feedback circuits which enhance the effect of boundaries in images; these work by adjacent cells either inhibiting or stimulating each other to make the contrast between black and white stand out more clearly. The result is that our eyes pick up edges and especially moving edges very well indeed. This has implications for screen design, making moving stimuli very noticeable, and for icon design in which clear boundaries become important.

More abstraction of image qualities is carried out in the next stage, image interpretation. The main point of image reception is that it is not just a photographic process; even at this early stage certain qualities of the physical image are being abstracted.

Image interpretation

Nerve impulses are transmitted from the eyes via the optic nerve to the optic cortex in the brain. Here images are translated into what we see. The whole process is still not completely understood; however, the basic

principles have been well investigated. Part of the optic cortex is organised in columns of nerve cells, each column being linked to a group of receptive cells in the retina. The cortex column cells have specialised roles for detection. Each cell type responds to a different primitive component in the image such as edges, corners, bars and gaps.

Depending on the pattern of nerve impulses coming from the retina cell group, one particular column cell will fire, transmitting the message that this part of the visual field has a particular shape (bar, edge, etc.) in it. By combination of many thousands of retinal cell groups an image can be built up as a composite of primitive features which define shapes, which in turn make up the complex pictures that we see. To supplement the shape outlines, the optical cortex gets more information from the retina. Retinal nerve cells are specialised for different receptive duties; some respond to colour, depending on its wavelength (red, blue and yellow), others detect movement, while some respond to the texture (such as rough or smooth) of surfaces in the image.

The optic cortex receives a mass of information coded in nerve impulses about different qualities of the image. The cortex then has to create visual meaning, the image we see, out of this information. It fulfils this task by referring to past records in memory, using an object–property matching process and reasoning about the objects within the image; for further detail the reader is referred to the work of the late David Marr (1982) who has described visual perception in detail. Marr demonstrated that we understand images by a series of processing steps; first objects are identified in terms of basic shape, then additional features are added including depth and perspective in the image to give a 2.5D sketch; further processing may then follow for a true perspective.

Object matching usually works very well but the result is that what we see is not what is there, rather it is our brain's interpretation of what is there based on memory and a mass of highly coded signals. Occasionally the process makes a mistake and we see a visual illusion.

Visual illusions use two tricks to fool the eye and brain; ambiguity and suggestion. Ambiguous images are ones which are open to two or more interpretations; different people will see different images because they have attached their own meaning to the picture. Some well known ambiguity illusions can be found in figure 2.7. Suggestion fools the eye by giving it a false clue in an image. The eye then supplies the missing information from memory to fit the clue, and creates an illusion of what is there. Only on closer examination does a contradiction become apparent. Suggestion can also work by supplying insufficient information in an image and then giving an extra clue verbally. People instantly see something in an image which beforehand they could not see, as illustrated in the Dalmatian illusion (figure 2.8). The implication of visual interpretation is that images are open to misinterpretation, because each person attaches his or her own

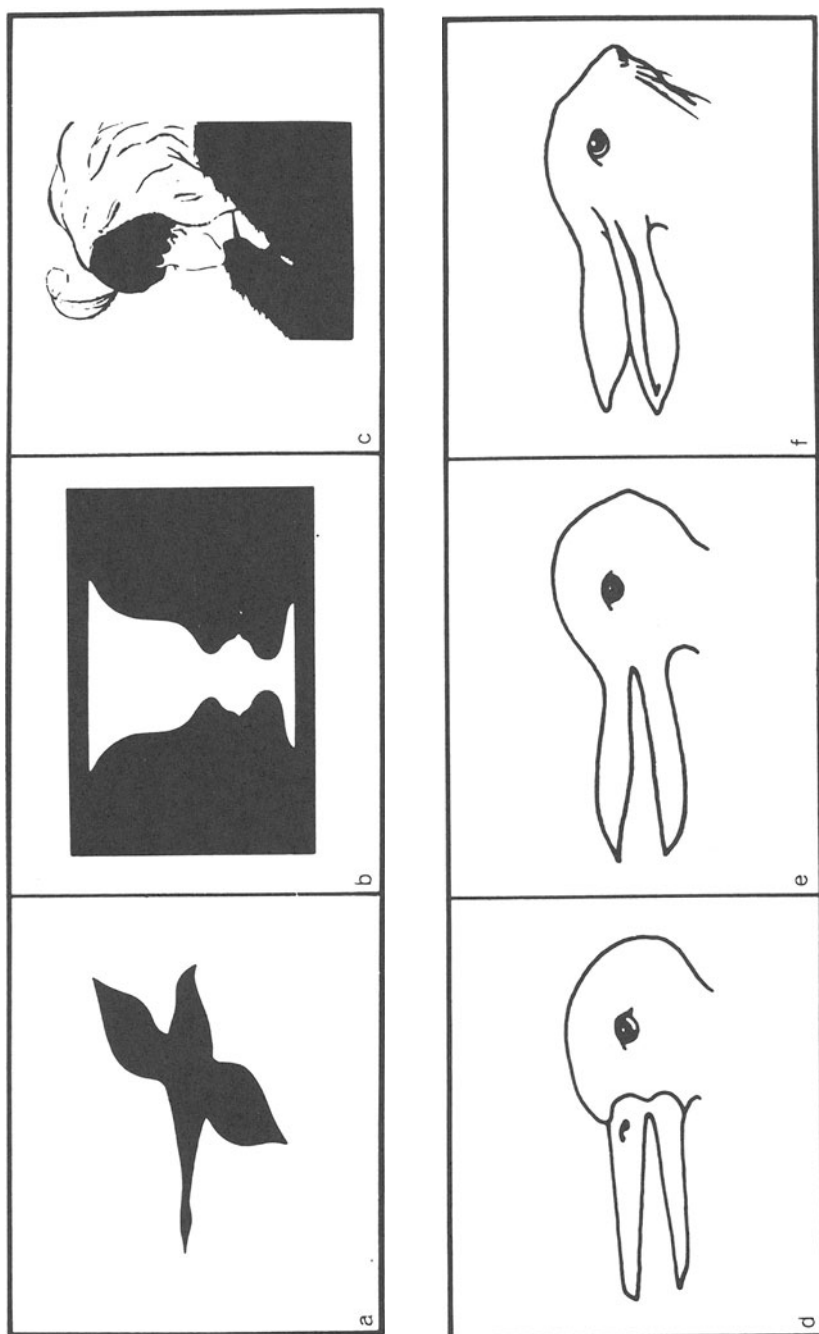


Figure 2.7 Ambiguity illusions: (a) hawk/goose; (b) vase/faces; (c) young lady/mother-in-law; (d), (e) and (f) duck/rabbit series.



Figure 2.8 The power of suggestion on interpretation. When prompted, most people see a Dalmation dog in the picture, some however insist it is a cow.

meaning to what is seen. As we shall see, icons, too, are open to many interpretations by different people. Correct interpretation of an iconic image can only be assured by testing its meaning.

2.3 Hearing

While vision is the dominant sense for human–computer communication at present, it is probable that hearing will assume at least equal importance in the future. Speech is the natural human communication medium and it would seem to be an appropriate method for computer control. Hearing involves the same set of problems as vision: reception of the stimulus, translating its properties into nerve impulses, and then attaching meaning to the nerve messages.

Sound is pressure waves in a gas. The air surrounding us is composed of gases, and sound is transmitted to us as a series of pressure waves in air. Sound waves have properties of frequency and intensity. Frequency is a measure of how close the sound waves are together and is recorded as the

number of waves arriving at a point per second, expressed as cycles per second or more often thousands of cycles per second—called kiloHertz. Sound frequency is usually described as *pitch*; the higher the frequency the higher the pitch of a sound.

Sound intensity is a measure of the energy in the sound waves, roughly how compressed the air molecules are in each wave. Intensity is related to the amplitude of sound, which is a measure of the sound wave energy at a particular frequency—see figure 2.9. We refer to intensity as loudness of a sound but, as with vision, what we hear does not always correlate with the physical measurements. Lower-frequency sound transmits more energy and is therefore technically louder. People, however, will reliably describe a high-pitch but physically low-intensity sound as being louder than a low-pitch high-energy sound. We react to our interpretation of sound, something quite different from the physics of what we receive.

Sounds are rarely composed of a single frequency; instead most sounds are a composite of waves at many different frequencies. Even a simple sound produced by a tuning fork has a main frequency and a series of extra, higher frequencies called *harmonics*. The tone of a sound, in the musical sense of the word, is produced by complex combinations of these harmonic frequencies. Complex wave forms can be resolved into a series of simpler waves by the process of Fourier analysis which describes the mathematical relationships between a complex wave and its components. The ear does a type of Fourier analysis on sounds and codes them as a series of frequencies and amplitudes corresponding to the wave's complex components.

As with the reception of light, considerable pre-processing occurs with hearing. The human ear is adapted to analyse complex sounds, and in particular speech. Speech is such a complex combination of sound waves that a graphical representation as shown in a spectrogram recording, illustrated in figure 2.10, looks like a complete blur. The ear has to detect all the separate frequency and amplitude components in speech.

Auditory pre-processing

Receptors in the inner ear show a similar specialisation to the optical system; some are tuned to fire for particular frequencies of sound, while others respond to the amplitude at a particular frequency. The ear acts as a series of semi-overlapping filters about a quarter of an octave wide. Nerve cells in each filter respond if part of the sound spectrum falls within their band; so a sound which is a composite of many frequencies is converted into a pattern of nerve impulses representing its various features. The filters have narrower band widths at lower frequencies with progressively wider bandwidths at higher frequencies, hence the ear is tuned to extract more information from lower-frequency sound. The frequency range for

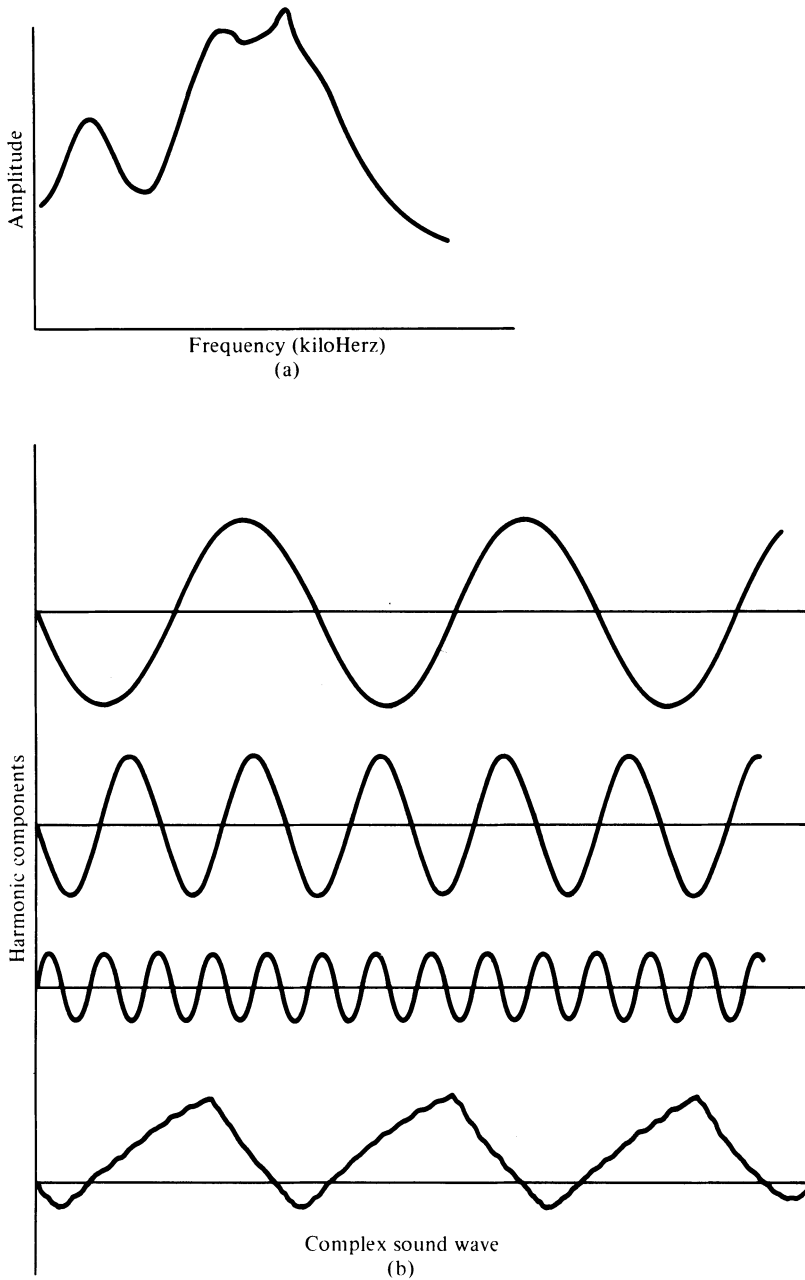


Figure 2.9 The frequency and amplitude of sound waves: (a) a simple sound wave showing the change of amplitude with frequency; (b) a complex wave decomposed into its harmonic components at different frequencies.

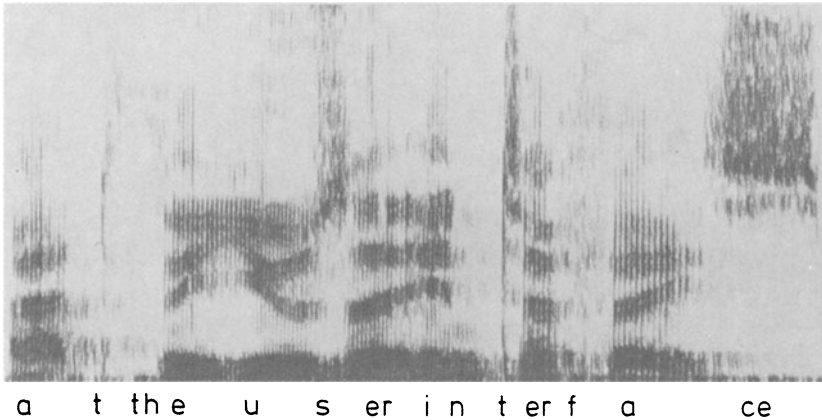


Figure 2.10 Sound spectrogram showing the continuous nature of speech, recorded as frequency over time.

deciphering speech is from 260 to 5600 Hz; however, the region of 2–3 kHz is most important. Telephones only transmit from 300 to 3000 Hz, yet we can hear speech quite adequately.

The ear has to extract certain sounds mixed in with background noise. The relationship of sounds to background noise is expressed as decibels (dB), a logarithmic ratio of the power of the sound:background noise, usually referred to as the *signal/noise ratio*. So not only does the ear have to be sensitive to the overall frequency range but it also has to resolve small-frequency components within the overall noise input. The key factors of auditory processing are:

- Frequency range for speech interpretation 260–5600 Hz, overall hearing range 200–10 000 Hz although this is individually variable
- Resolution capable of telling frequency components one-quarter of an octave apart
- Temporal resolution of sounds separated by 5–15 milliseconds (ms)
- Amplitude resolution of 1 dB in peaks of sound

Interpretation of sound

The most important aspect of sound from a human point of view is language. Sound interpretation is integrally linked with language understanding, both functions being carried out in the auditory cortex of the brain. To interpret sound the auditory system has to classify the input into three categories: noise and unimportant sounds which can be ignored; significant noise, that is sounds which are important and have meaning

attached to them such as a dog's bark; and meaningful utterances composing language.

The hearing system, like vision, makes use of past experience when interpreting input. Spoken language is full of mispronounced words, unfinished sentences and interruptions; furthermore, it happens quickly so the interpretation mechanism has to keep pace with the input. Speech rates are in the range of 160–220 words per minute, so interpretation has to be rapid.

Language recognition from speech has to start by discovering the basic sound units of language called phonemes. These sounds can then be matched to the basic units of written language, called morphemes which correspond approximately to syllables, suffixes, prefixes, etc. and thereby words. Phonemes describe all the possible sounds in a language. Some languages possess many sounds; for instance, Norwegian has 24 different vowel sounds alone, while 40 phonemes make up all the vowel and consonant sounds in English. Phonemes may also differ considerably from the written language, as in English plural nouns which, although written with an 's/es' suffix have two different sounds, a 'z' as in *hens, fens* and 's' as in *books, locks*.

Interpretation, however, does not use phonemes alone, it is a layered and integrated approach in which the brain makes use of language syntax (the grammar), semantics (the meaning of words and sentences), and pragmatics (knowledge of the context of communication), to decipher communication.

Speech does not appear as a sequence of conveniently separated phoneme sounds but as a continuous band of sound throughout a phrase or sentence. Our ears extract most information from the lower frequencies where resolution is better, but temporal patterns in higher frequencies are also important. Simple template matching of sound spectrograms to phonemes is unsatisfactory because of the problem of finding word boundaries; in addition, a wide variety of physical sounds can be generated for one phoneme by different speeds of speech, different dialects and speech inaccuracies. It is the knowledge of language syntax and semantics which enables us to break the continuous speech into discrete phonemes and words. People supply a significant amount of what they hear on the basis of expectancy. This can be demonstrated by experiments asking people to identify a sound masked by a cough in the middle of a sentence. Most subjects reply that no sound is missing. Further evidence of verbal suggestion is demonstrated by an experiment in which one word 'eel' was heard as four different words depending on the sentence context:

It was found that the eel was on the axle
It was found that the eel was on the shoe
It was found that the eel was on the table
It was found that the eel was on the orange

The sound 'eel' was heard as wheel, heel, eel and peel respectively in the four sentences (Warren and Warren, 1970). Speech recognition also suffers from illusions in a similar manner to the visual system. In speech the timing of perception is more critical and as a result the tolerance of speech-interpretation mistakes is higher; consequently illusions in speech are not referred to as such.

Memory plays a crucial role in both vision and hearing; consequently the role of perception, in the sense of receiving information, and cognition, in the sense of understanding and using external information, cannot be meaningfully separated. This leads to investigation of how memory works and how it is used in the processes of understanding and reasoning.

2.4 Learning and Memory

Human memory comes in two varieties: short-term working memory and long-term permanent storage. The information-processing model will be used to place memory in the perspective of perception and cognition.

According to the model, each perceptual sense has a processor and associated short-term memory. These memories form the input and output buffers of the human system, storing abstract images in visual short-term memory and sounds in auditory short-term memory. Each memory is associated with a sensory processor. The sensory processors analyse the contents of their memories and pass the resulting information to the cognitive processor for identification of the sensory input. The overall schema of the model human information processor is illustrated in figure 2.11.

The capacity of sensory short-term memory is not clear, but for vision it must be at least the contents of one visual field. The contents decay rapidly in about 100 milliseconds and are continually overwritten by new input; for an illustration, when you close your eyes the visual image vanishes quickly. The visual input buffer has to be overwritten because the quantity of data in an image is vast and images change continually; consequently storing even a few images would take a vast amount of memory. The auditory input buffer, also referred to as echoic memory, may contain several phonemes' worth of sound because no one millisecond of sound contains enough information for correct language identification. The contents of echoic memory probably last for up to 1 second before they are lost.

The contents of visual and auditory short-term memories are in an abstract form after sensory processing, although no meaning has been attached to the input at this stage. Meaning is generated when information in the input short-term memories is passed on to the central cognitive short-term memory for interpretation. The cognitive processor is thought to be responsible for object identification. This is effected by matching the

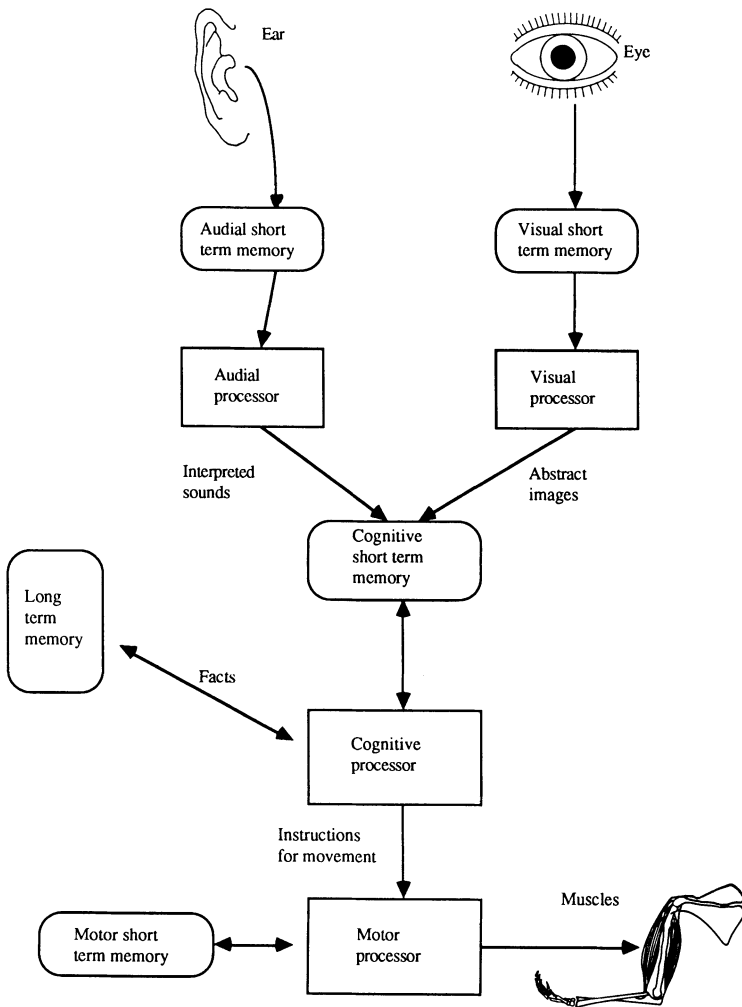


Figure 2.11 Information Processing Model of human perception and cognition (after Card et al., 1983).

incoming information with past experience and then attaching semantic meaning to the image or sound. To complete the model, the cognitive processor has an associated short-term memory which is used for storing temporary working information. The information may have come from the sensory processors or may have been retrieved from long-term memory.

The cognitive processor performs most of the actions which are considered in everyday language to be thinking. The results of thinking are

either placed back in short-term memory, or may be stored in long-term memory, or may be passed on to the motor processor to elicit behaviour by operating muscles. The motor processor is responsible for controlling actions by muscle movements which create human responses and behaviour, such as running, talking, pointing, etc. The motor processor has its own short-term memory to store input from the cognitive processor. Its output is sent down the peripheral nervous system, which forms the body's data communications network to the muscles. Speech output is a special case which requires a separate output processor and buffer of its own. Evidence indicates that approximately 2 seconds' worth of speech can be held in the buffer which allows words to be assembled in a sequence for rapid output.

The information-processing model provides an outline description of the cognitive apparatus, although the whole system is known to be more complex. At this point, the important distinction to make is between the roles of short-term and long-term memory.

2.4.1 Short-term Memory

Short-term memory (STM) is the human equivalent of computer RAM memory, in other words the working memory of the central processor. In contrast to computers, human short-term memory loses its contents unless it is refreshed every 200 ms; however, the read/write access time, about 70 ms, is quite quick so information can be held in STM by continual rewriting.

According to the information processing model, short-term memory has to store information from many sources, hence it may seem strange that experimental evidence indicates that it has a very limited capacity. In an influential paper, Miller (1956) summarised experiments which placed a limit on short-term memory of seven items plus or minus two. Items were not stored as in computer memory 'bytes' but in 'chunks' of information. These can vary from simple characters and numerals to complex abstract concepts and images. The secret of expanding the limited storage in STM is to abstract qualities from the basic information and store the abstraction instead.

This concept is best understood by example. Telephone codes may be given in an unordered fashion, such as 0612363311; such large numbers are difficult to assimilate and remember, but break the number up into smaller units and memorisation is easier, for example, 061-236-3311. The effect is to suggest a chunking strategy to the reader. Instead of storing ten separate digits, the number groups can be stored as whole chunks, reducing the storage required from ten chunks to three. The more order which can be imposed on the raw data, the better the chunking. To convince yourself of the point try to memorise the following quickly:

832751984221 — accurate recall would be unusual

061–236–3311 — should present no problems

246

357

81012

91113

} should also be recalled without error
once the pattern has been seen

The second and third number sequences have order within them that promotes chunking. What has been stored is some quality of the data which can be used to reconstruct it: in the latter case, the algorithm of even/odd triplets in an ascending numeric series.

In summary, the important features of short-term memory are:

- Rapid read/write access time—70 ms
- Memory decays quickly—200 ms unless refreshed
- Capacity is limited to 7 ± 2 chunks
- Storage capacity can be increased by abstraction qualities of raw information

More recent research has shown that the information-processing model is a little simplistic (see Hitch, 1987). STM has at least two sub-systems; one deals with language-based data while the other deals with visio-spatial information. The linguistic sub-system functions as a list but access is like a hybrid LIFO (last in first out) queue. We tend to remember the last and first few items in the list and forget the middle. Storage and retrieval are generally sequential. The whole short-term system, called *working memory*, is controlled by an executive, similar in concept to the cognitive processor. This more elaborate model helps explain how temporary memory for visual and textual information differs and how interference during memorisation impairs retention of information. In the latter case the executive appears to be distracted during the process of storing and refreshing the contents of working memory.

Some key features of working memory are:

- Distraction causes forgetting of recently learned material. Even a small number of simple chunks of information are lost within 20 seconds if there is distraction during input
- Other inputs impair recall. Supplying irrelevant material during input to working memory makes recall worse
- Very similar inputs impair recall. Supplying closely related items during memorisation makes recall worse
- Immediate memory for details in complex images is poor
- Recall of items is better if both the word for and a picture of the item are presented together, compared with the image or word in isolation
- People remember in the short term (<30 s) by scanning back along the input—thus last in first out

The consequences of working memory lead to some general guidelines:

- Minimise distraction during tasks and memorisation
- Beware of overloading short-term memory, both in terms of quantity of information and time span of retention
- Structuring (chunking) information helps memorisation
- Images are helpful but need to be accompanied by text

The central role that short-term memory plays within computer interface operation will become apparent in chapter 3 on task design. Short-term memory limits our ability to process information during tasks. Its counterpart, long-term memory, is important in storing the knowledge which we use to help us understand and perform tasks.

2.4.2 Long-term Memory

Long-term memory is the main file store of the human system. It has a near infinite capacity as no-one has been able to demonstrate an upper limit on what we can remember. Memory failure appears to be a problem of not retrieving what is already inside our memory.

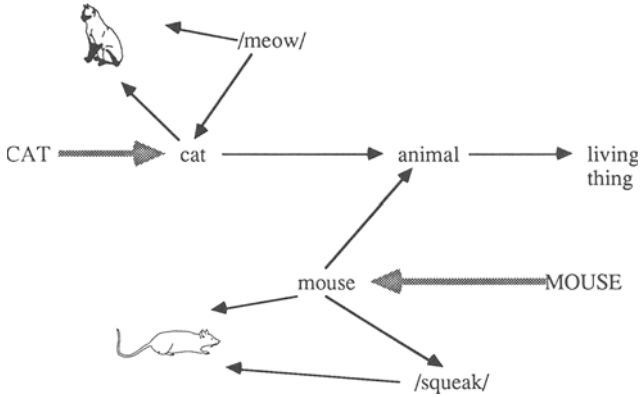
Retrieval of facts from memory can be remarkably fast, especially for frequently used items and procedures. Retrieval time for information used less frequently varies; it can be quick, but may be slow especially for older people. Retrieval according to the information-processing model is a function of the cognitive processor, but in reality the process must be more complex. Often, remembering a fact is not instantaneous; instead it comes back some minutes after the original effort to retrieve it. During the intervening time attention will have been devoted to other matters, hence it appears that a background memory processor must be invoked to effect difficult long-term memory searches.

The basic organisation of this memory is thought to be semantic, that is, data is stored in terms of linguistically based concepts linked together in a highly developed network. An over-simplified analogy is to consider long-term memory as a sophisticated network type of database with access paths following a line of associative pointers to the information. A semantic network model, as depicted in figure 2.12a, is not the whole story. Memory also has categories which contain many related items, and the network may act as a link to these categories. However, there are probably two types of access, one chain of semantic associative pointers and a more direct access mechanism via images. This gives rise to the possibility of two types of human memory, associative and analogue, the former storing concepts while the latter stores more concrete objects such as images and sounds.

Memory mechanisms

How memory works in the physical dimension is still a subject of research. One hypothesis predicts that it is a chemical process of making network

(a) Semantic network model of memory. Objects are associated in a network of classification and attributes. The image of the object is not stored in a photographic form, rather a representation is generated from the network of interconnected labels which describe it.



(b) Category model. Objects are held within categories which have descriptive tags for recall. Individual objects are not directly addressable, instead they are recalled by list searching the category contents.

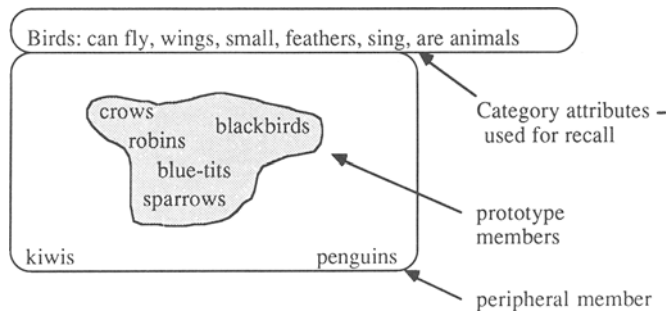


Figure 2.12 Organisation of memory: semantic network and category models.

links between nerve cells, and that the act of remembering re-creates the links. Computer simulations of learning, which may be regarded as a form of making new memories, have shown that within network models, human learning can be mimicked by complex algorithms which control how associations are formed between nodes in the network. Certain algorithms can form new network pathways from inputs to outputs by altering weights on connections. These weights mimic the synaptic structure of nerves and control whether the influence on the next node will be excitatory or inhibitory. One such algorithm is Hebb association which states that if two

adjacent nodes are activated together then the weights should be changed to increase their association. This creates associative learning as illustrated in figure 2.13. Simultaneous firing by parts of input neurons has the effect of strengthening their connections to the output neuron.

The models work by iterative cycling of the algorithms around the network until a stable pattern of associations emerges. Interpretation of these patterns requires a human observer, but in some cases it appears that new meaningful associations can be generated. In one example a network input representing royal family trees in a parent/children form created new outputs which described kings and queens in terms of brothers and sisters. This research, called *parallel distributed processing* (Rumelhart and McClelland, 1987) may form a credible model of human memory as there is some evidence that human nerve cell networks change their connective properties during learning. So memorisation and learning may be by formation of complex pathways in neural networks. Forgetting, on the

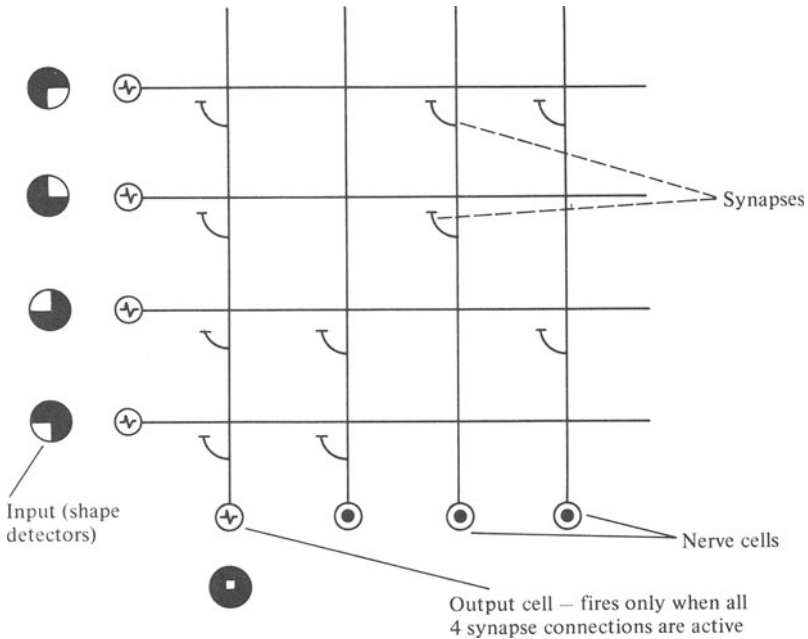


Figure 2.13 Memory schema: possible neural organisation in visual perception. The input comes from edge detectors; connections in the matrix can then detect different firing patterns among the input nerve cells. In the example the cell connected to all four inputs fires when a square is found. Firing in adjacent cells could make the synaptic contacts with output cells stronger.

other hand, happens when the links decay with age or were poorly formed in the first place.

Memorisation is usually an effortful process. There are various methods of memorisation, the simplest being rote learning in which information is committed to memory as a list with few associations between individual items. An example of rote learning is the practice of learning tables of numbers by heart. However, most learning is by association, in which facts are linked together to provide an access path. There is experimental evidence that the greater the number of separate access paths, or the more often an access path is used, the easier a fact is to remember. The depth of processing in terms of elaborate reasoning carried out during memorisation also helps recall in the future.

Organisation of memory

The organisation of human memory is far from clear, although most evidence favours the view that all storage is finally of the semantic associative kind, with two different, linguistic and visual, access mechanisms. There are two types of knowledge of importance for human-computer interaction. First is categorical knowledge, that is objects and their associations. In this case memory may be organised in categories and the access mechanism finds the category although, as indicated by experimental evidence, not the members within a category. There is evidence that we organise the world not into discrete non-overlapping categories but in a more fuzzy manner, with core and peripheral members. To illustrate, most people have an idealised concept of a bird. A robin fits this core or prototypical image, having the properties: round, feathered, sings, lays eggs, etc. In contrast, a penguin is a more peripheral member of birds because it does not share all the attributes of the prototype image and it has additional non-standard attributes, for example, it swims and cannot fly. The concept is illustrated in figure 2.12b. Retrieval is more rapid and accurate for core items in categories and slower for peripheral items.

The second type is knowledge about actions and how to do things. This is held in two different forms; declarative or rule based knowledge and procedural knowledge. When we start out knowing little about a subject, we acquire fragments of declarative knowledges as rules and mini-procedures. This knowledge, however, is not organised, so to carry out a task we have to reason with declarative knowledge fragments and compose them into a plan of action. This process, often described as 'figuring it out', involves considerable effort. That effort is the demand on short-term memory as we organise the knowledge fragments. As people become more familiar with a task, these fragments become compiled into procedures that can then be run automatically. Hence when we know how to do a task we simply call the procedural knowledge of how to perform it automatically. This is easy because the short-term memory load has been avoided.

Another view of memory is based on how information is stored. One type is episodic memory; here associations are made in a context. The other type is semantic memory in which associations are stored in an organised manner. The former requires less effort, for example, we remember objects on a desk which provides the context. Semantic memory, however, requires understanding of what is being stored rather than a loose association such as spatial proximity and temporal co-occurrence. In spite of this, episodic memory has its uses. It is a powerful means of recall especially when visual cues can be given. The icon-based desktop metaphor of Apple and Xerox workstations uses episodic memory to help us remember and understand the system as objects on a desktop.

Storing information in long-term memory is generally linked to understanding facts. This is demonstrated by the way people reconstruct information from memory. Storage of data on every object of interest would swamp even the large capacity of human memory, consequently associations are stored with a limited amount of basic data. To illustrate the point, try to find out in which compass direction you are facing while you read this book. This may be an easy task if you know your room faces a particular direction; a more likely scenario is that you will establish the direction either by reasoning based on where the sun rises and sets or by using geographic knowledge of landmarks which you can see. You can synthesise knowledge from associations between memorised facts rather than storing each fact individually. By storing links between facts we can memorise a large number of facts and reconstruct even more information by processing those links in new situations. The reasoning process which happened during memorisation is important for recall. For instance, you may have established the direction from the link between sunsets-west and the observation that you know where the sun sets in the view from your window.

Memorisation techniques

Formation of access paths can be helped by memorisation techniques. Perhaps the most famous of these was invented by Solomides, an ancient Greek poet. His technique was to associate information with spatial features of a house; so the first part of a speech was linked with the entrance hall, the middle part with the living room, and the end with a bedroom, etc. This technique formed more associative links during memorisation and possibly exploited the visual access path to memory. Other techniques involve coding extra semantic cues in memory pathways by learning additional associations with the object to be retrieved. Examples are keywords, peg words, mnemonics, similes and acronyms.

Memorisation fails because an access path either decays through lack of use or was poorly constructed in the first place. Similar facts can interfere

with recall, so well recognised access paths which are sufficiently distinct from others are helpful in preventing recall errors. Distractions during the memorisation process also cause recall errors as the access path is liable to be incomplete. So if attention is diverted during memorisation, for instance by a noisy environment, memory performance will suffer.

Memory is one of the critical limiting factors of human information processing which affects interface design in many ways. Interface design should strive to reduce the amount which has to be learned; and when learning is inevitable, recall should be helped by memory cues. We deal with the complexity of the world by ordering and classifying it. The interface designer should support this process by imposing structure on a design, one of the basic HCI principles. We understand and memorise complex information by breaking the complexity down into simpler components using a hierarchical approach. Complex objects are remembered, and hence understood, by storing facts which compose and describe the object at various levels, in combination with the access path of associations by which we analysed and understood the object in the first place. The more structure and categorisation we can put into a body of information, the easier it is to learn.

A second HCI principle which is important for memorisation and learning is consistency. The more consistent something is, the easier it is to perceive patterns within it and hence to learn its structure and characteristics. Humans are good pattern recognition and association machines; anything which helps to establish a pattern will help to reduce the memory burden. A summary of the important features of long-term memory are:

- Effectiveness of recall is correlated with the depth of processing on input, that is, the effort put into memorisation
 - Recall is helped by unique cues and the distinctiveness of the item in memory in relation to other items stored in the same context
 - Recall is hindered if distracting and irrelevant material is presented during memorisation
 - Recall suffers if one cue is used for many different objects (cue overload)
 - Recall is better for pictorially presented material and for text presented with pictures than for text alone
 - Recall is better if the context of remembering fits the context of memorisation (episodic match)
 - Similar items are more likely to be grouped in categories
 - Within categories, prototypical items are easy to memorise and recall
- General guidelines can be derived to help memorisation and recall; however, as with short-term memory, care must be exercised in applying these guidelines. General advice does not always fit into specific contexts.
- Memorisation can be helped by enriching the information during learning. Reasoning and understanding what is being remembered help

- Structuring information helps reasoning and creates extra links to retrieve items
- Techniques can be used to add extra recall cues, for example, keywords, spatial memorisation, etc.
- Visual presentation with text helps learning and recall
- Consistency of associations creates better contexts for memorisation and recall

2.5 Thinking and Problem Solving

Thinking, reasoning, and problem solving are all human mental activities which process data derived from our senses and long-term memory. Problem solving is something that we do every day of our lives when we come up against something unexpected. It may be defined as ‘the combination of existing ideas to form a new combination of ideas’. An alternative view focuses on the cause. Problems arise when there is a discrepancy between a desired state of affairs and the current state of affairs and there is no obvious method to change the state.

Problem solving progresses through several stages; the names of stages vary between authorities on the subject, so the following scheme is a generalisation:

- (a) Preparation or formulation: the goal state is defined and necessary information for a solution is gathered.
- (b) Incubation or searching: anticipated solutions are developed, tested, and possibly rejected, leading to more information gathering and development of alternative hypotheses.
- (c) Inspiration: the correct solution is realised.
- (d) Verification: the solution is checked out to ensure it meets the goals and is consistent with the information available.

To illustrate how problem solving may work, another model will be employed. The Goals Operators Methods Selection rules (GOMS) model of Card *et al.* (1983) owes its heritage to the General Problem Solver model of Newell and Simon (1972).

Problem-solving models

The GOMS model is composed of a set of goals and sub-goals organised in a conceptual problem container, called the problem space. During the problem-searching phase, goals are broken down into a sub-goal network; searching then proceeds by traversing the network and testing hypotheses at each node. At each sub-goal node data is read into short-term memory,

evaluated and then stored back into long-term memory as searching progresses to the next sub-goal.

Eventually, if the search network has been well constructed and all the facts are available to be evaluated, and the sub-goals pass the tests, the final solution node is reached, resulting in the problem solution. This operation is the familiar reasoning strategy of problem solving by steps, namely if X is A, then Y is probably B, which means that Z must be true . . . etc. However, not all problems can be approached in such a sequential manner.

Other components of the model are operators which describe the sequence of actions necessary to reach the goal and methods which control the strategy or approach to the problem. Operators are controlled by selection rules, that is, production systems which invoke an operation. Facts are evaluated to give results either proving or disproving a sub-goal. According to the results, the goal network may be re-organised as new hypotheses are introduced and old ones discarded.

Methods describe how the network is formulated and traversed; essentially they are the problem-solving strategy. Humans use a variety of strategies, some of which they can articulate but some appear to be unconscious as in solutions which 'come in a flash'. This leads to difficulties when analysing human problem solving. The accepted method is protocol analysis, basically thinking out aloud, by asking the subject to verbalise the problem-solving steps and procedures. Unfortunately humans are often unaware of their own procedures in detail, with the consequence that steps are omitted. Reasoning analysis, therefore, poses problems for knowledge acquisition and problem-solving analysis in expert systems.

There are different strategies or methods which can be applied to reasoning. One is known as *forward chaining*, in which facts are known allowing the 'IF(condition)' part of an If-condition-Then-action production system to be evaluated. The chain progresses forward to an action or the next IF test. The mirror image is *backward chaining* when we have facts relating to the consequences (action) part and reasoning progresses backwards to establish the IF condition which is consistent with the observed facts. People use both methods interchangeably.

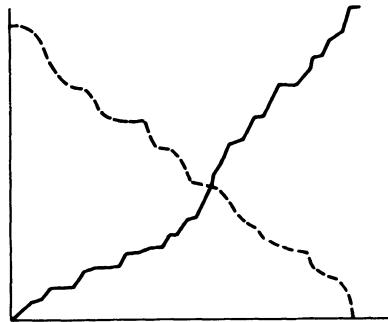
Another problem-solving method is *inductive reasoning*. This case is similar to classification; by observation of facts we conclude a new fact which describes the initial assertions. Faced with a menagerie full of cows, lions, giraffes and bears, the observation may be made that they all have four legs, leading to the conclusion that animals are quadrupedal.

Various other strategies of problem solving are used by people, some of which have been incorporated into the semantics of databases, for example, aggregation of properties to define an object, inheritance of properties in a classification scheme of objects. Success in problem solving

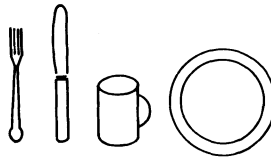
can often depend on using novel strategies, such as visualising the problem in spatial terms or treating it mathematically, as shown in figure 2.14. People are naturally conservative in their approach to problem solving, and adopt the methods they are used to.

Mental models

Another common reasoning strategy is *deduction*. Deductive reasoning starts with assertions and discovers new facts by logically examining the



(a)



(b)

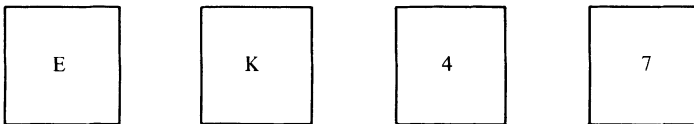
Figure 2.14 Two methods of problem solving. (a) Visualisation of the Buddhist monk problem. The problem is: a monk climbs a mountain path starting at dawn, stopping for rests on the way up and arriving just before sunset. The next day he descends by the same path, again stopping for rests but going faster than on the way up. Demonstrate that there is a point on the path which the monk will occupy at exactly the same time of day on both the up and down journeys. (b) Visual mental model of the problem to order 'the fork is on the left of the knife, the plate is to the right of the cup, and the knife and plate are not adjacent.'

relationships or properties which the assertions describe. This is associative, or syllogistic reasoning of the style:

All animals with wings can fly	}	propositions or known facts
Bats are animals		
Bats have wings		
Therefore bats can fly		a new conclusion based on the propositions

The procedure is to pattern-match items and the truth conditions attached to them, from which new combinations of facts can be made. Logicians have formalised this process as propositional calculus and its more sophisticated brother, predicate calculus. People, however, do not obey these formalisms. While we reason well in terms of positive association, when negative terms are introduced our reasoning becomes illogical. Take the following problem, which is a classic in psychology:

You are given four cards: on one side there is a number and on the other a letter. A rule states that if there is a vowel on one side then there must be an even number on the other. Which cards should be turned over to prove the rule true or false?



Most people go for card E and 4. Logically this is not correct because the rule states a vowel–even number link and not the converse. Finding a consonant on the reverse of 4 proves nothing. The correct answer is E and 7.

However if the problem is restated in more concrete terms, performance improves. Test yourself with the following. There are four invoices on a desk. Invoices are marked pro-forma and normal payment. Pro-forma invoices must be paid and stamped on the reverse side with ‘Payment Received’ before goods can be dispatched. The four invoices are face-up pro-forma, face-up normal payment, face-down unstamped, face-down stamped. The question is the same as before: prove the rule that pro-formas must be stamped. Performance this time should be better (pro-forma and unstamped is the solution), although the underlying logical properties of the problem are identical.

It appears that the content and context of a problem are more important than underlying logical structure and that reasoning in abstract terms is more difficult than in concrete examples. The important consequence of this is that we transfer our knowledge about context and content between problems rather than the underlying logical structure. This has implications for task design because unfortunately knowing one task does not help learning of another task with the same underlying logical structure. Instead, context influences our decisions which may result in the wrong method being applied to a problem because superficially it appears to be similar to a previous one.

Human and machine deduction are very different. Human reasoning uses logic loosely and backs it up with associations, that is, knowledge about the objects in the problem. To illustrate the point, consider the assertions:

Some animals with wings can fly
 Birds have wings
 Therefore birds can fly

The conclusion may be regarded as valid but it is not logically so because the assertion does not state that all animals with wings can fly. We may also refute the argument from our knowledge that penguins have wings but cannot fly. We appear to construct mental models of things in terms of propositions or truths which we hold to be true on the basis of memory. These truths are then used in reasoning rather than logical examination of the problem in detail. The explanation of cognitive processes by mental models has been advocated by Johnson-Laird (1983) and this work has had a wide influence on cognitive psychology.

Mental models help explain observable phenomena about human mental abilities such as our inability to reason logically in some situations. Human ability to reason logically may be limited by working memory because to solve problems several associations have to be held in working memory. Consider reasoning about the following:

Some Artists are Brokers
 All Brokers are Consultants

We form a model of the propositions symbolically

A = B	B = C	where () denotes an independent
A = B	B = C	existence and = is a link equating
(A) (B)	B = C	the objects

Reasoning then proceeds by substitution to create

A = B
 (A) (B)
 A = C
 (A) (C)

We conclude that some artists are also consultants.

This is an easy example where the number of concepts does not exceed working memory limits but, when the number of terms increases, more than one mental model can be constructed for a set of propositions; and the relationships to be held in working memory increase. Furthermore, when negative terms are added this militates against the positive pattern-matching process. Not surprisingly we reason poorly with complex logical relationships involving negation. To prove the point, consider the following:

No Brokers are Artists
 Some Brokers are Consultants
 Are any Artists also Consultants?

More than one conclusion appears to be possible because more than one mental model can be constructed. In another case of No A are B and No B are C, there are two conclusions:

There are three disjunct sets A, B, and C
 Set A and C may however be related, even though A,B and B,C are not

Mental models may be either physical or conceptual. Physical models describe the relationships of objects in the real world in terms of spatial distribution of events in time. Physical models may be visualised in a spatial manner, especially if the problem involves spatial reasoning, such as the fork is on the left of the knife; the plate is on the right-hand side of the knife. Conceptual models come in different manifestations. There is the surface linguistic expression, then an internal mental language which, although linguistically based, represents a further abstraction. Conceptual models are a type of internal mental language representing truth values of relationships with which we can reason. The form of mental models differs between people and depends on individual cognitive styles. Mental models are important in creating human-computer interfaces, and this theme is revisited in chapter 3. The main point to note is that mental models should be based on people's experience, that is, the truths which they may be expected to hold.

Skills and errors

We problem solve in two modes. If we know little about the problem we use previous knowledge and general rules of thumb or heuristics. This attentional reasoning is a difficult process which makes heavy demands on working memory, as we form the problem-solving network. After experience, problem solutions are stored in memory and the correct calling conditions then invoke automatic procedures which consume less effort. People tend to minimise mental effort whenever possible so there is a natural tendency to use automatic procedures if possible and to automate new procedures with practice. Use of automatic behaviour presents a dilemma in matching calling conditions to the correct procedures. In such situations we make mistakes. Errors in problem-solving tasks can be classified as 'slips' which are errors in carrying out a correct sequence of actions and 'mistakes' when the plan of action was misconceived in the first place. Slips are probably caused by a distraction or failure in attention so that a step is missed out or not completed. True mistakes, however, are a failure in matching the correct procedure to the problem.

People are generally good at heuristic reasoning and this ability marks us apart from even the most sophisticated artificial intelligence machine. Ironically, however, when we are under pressure this ability often deserts us; we revert to automatic procedures which may well be inappropriate. There is evidence to suggest that people select procedures on the basis of frequency of use if environment cues do not identify the correct memory exactly. This frequency gambling can lead to unfortunate consequences, some of which have been manifest in accidents in nuclear power stations.

Acquisition of skill is by learning, the process of acquiring new memories for behavioural sequences and mental procedures of problem solving. Skill learning is subject to a law of diminishing returns known as the power law of practice; the time taken to complete a task plotted against the practice time forms a straight line on a log-log plot. The effect is that more practice yields an increasingly small improvement in performance. The power law can be formalised:

$$T = c + a(P + d)^{-b}$$

where c = near maximum speed (asymptotic)

T = task completion time

a = initial speed

P = practice time

b = number of trials

d = possible number of trials before measurement.

Acquisition of skill is influenced by factors which also affected memorisation. Frequent, regular learning sessions help skill acquisition whereas gaps

without practice help forgetting; positive feedback during task performance helps automation, as does presenting a clear model to the task and making the task steps easily recognisable. Redundant feedback only confuses. Skill learning is improved by use of context-dependent learning; this is also important in binding activation of the skilled procedure in the correct circumstances. Speed and ease of automation of task sequences are correlated with number of steps within a task.

Skills and automatic processing are important mechanisms for the human machine. It enables parallel processing to occur by reducing the need for attention to external stimuli and the load on short-term memory. The penalty we pay is that sometimes our automatic procedures run in the wrong circumstances, in the face of environmental cues which obviously contradict the course of action.

The implications of human reasoning for interface designers are that tasks should be structured to help users solve problems. This can be done by constructing a clear mental model for the user which invokes appropriate parts of the user's experience. The GOMS model may be used as a framework for design; breaking the problem down into goals, enabling operators to test the goals and providing an overall method for approaching the problem. An over-rigid definition of problems in systems may be counter-productive as humans use many different methods to solve problems. In spite of good analysis, the designer may not choose the correct one. Hence in decision support tasks, making the goals and operators explicit could be advisable, but choice of the method should be left to the user. In expert systems, the analyst is recording a problem domain and strategies for finding solutions; in this case the goals, operators, rules and methods have to be specified. Before moving on to the implications of human problem solving for human-computer interaction, it is worth summarising the salient features of human reasoning:

- We reason by applying procedures to memorised facts and environmental information
- Problems are formulated as mental models of associations and truth values, possibly organised into spatial terms
- There are a variety of different procedures which can be applied to problem solving including backward and forward chaining, syllogistic reasoning and classification
- Human reasoning is not strictly logical, rather it is a comparison against a series of propositions which make up a mental model
- Reasoning is heuristic in situations where little is known about the problem. Heuristic reasoning requires considerable effort
- Experience leads to the results of reasoning being stored as automatic procedures
- Automatic procedures have calling conditions. Mismatch of calling conditions and procedures can cause mistakes

So far, storage of data and processing, memory and reasoning in the human machine have been examined. The next element of the human machine is control; how all the conflicting demands of problem solving, memorisation and recall, and sensory input are resolved. In computer terms this is a scheduling problem; the human equivalent is attention.

2.6 Control of Human Information Processing

The information-processing model gives a picture of a sequential machine with a bottleneck at the cognitive processor and its short-term memory. Even though we may be sequential to an extent in our reasoning processes, the human machine is capable of considerable multi-tasking. The control of activity is partly automatic and therefore unconscious, although some control is in the realm of our conscious. This we refer to as paying attention.

2.6.1 Attention

From the information-processing model it should be apparent that there are several input/output channels competing for the resources of the cognitive processor and its short-term memory. Inputs from the visual and auditory systems compete with other senses which have not been reviewed, such as touch, smell and pain. In addition, the cognitive processor has to find time to access memory and control output to the motor processor and speech buffer.

The fact that we are basically sequential machines should be apparent from our poor ability to do two or more mental tasks concurrently. Try reading a newspaper and listening to the radio at the same time; either the radio or the newsprint will be remembered but not both. Attention is selective, the best we can do is to time-slice between channels so that we remember part of what the radio announcer said and a few things from the newspaper article. In spite of our sequential attention we do have considerable capacity for concurrent processing. We have already encountered background memory tasks, and parallel processing of input; in addition we also do certain actions automatically, for instance driving a car while holding a conversation. These automatic actions are more usually called skills. Action sequences for skills are stored in long-term memory and subsequently accessed for output to the motor processor as instructions for an activity.

To complete all its tasks the human machine must have more than one processor running concurrently. When driving a car and talking, the motor processor will be controlling the leg and arm muscles for steering and braking; the speech processor will be controlling the larynx to form speech,

while the cognitive processor divides its attention between monitoring the senses for road traffic and listening to what has been said. Such complexity appears to strain the resources of the information-processing model to its limits. Recently more complex and flexible models of human mental activity have been proposed which account for more concurrency in human mental activity by envisaging a cooperative system of parallel processors. Attention in the form of a system monitor must still have a key role.

Although some parallel processing undoubtedly occurs, there is a limiting sequential bottleneck in cognitive processing. Resource rationing has to occur and like a computer this is controlled by scheduling with interrupts for important events. If little of interest is happening in the environment we pay little attention to sensory input, as may happen when we are lost deep in thought. The instant something unexpected happens, for example a loud noise, our attention is immediately switched to the sensory input. The visual or auditory processors effectively put an interrupt on the cognitive processor. The input processors are continually competing for the cognitive processors' attention in this manner. In this battle our attentional apparatus is finely tuned to ignore constant states and pick up changes in the environment.

Unfortunately the human ability to ignore the steady state in the environment can lead to poor performance in monitoring tasks. If we have to concentrate on one channel containing input with little variation, there is a natural tendency to ignore changes and for attention to wander as the cognitive processor polls other channels. Even worse, in long monitoring tasks fatigue may set in, causing the cognitive processor to miss significant events in environment. Distractions are very effective at diverting attention, particularly if the information is irrelevant to the task in hand. This probably occurs because the attention controller naturally polls all input and enforced attention tries to over-ride this mechanism with the undesirable effect of making people more sensitive to distracting signals.

Attention is influenced by the difficulty of the task attention is being paid to, by the distraction in the environment, and motivation of the individual. More difficult tasks hold attention better than mundane boring ones, which explains why most people will read a good book without degraded attention but watching a stationary blip on a radar screen soon becomes boring and performance suffers. Motivation is the internal will of an individual to do something, which can be influenced by physiological factors (such as hunger), psychological factors (such as fear) and sociological matters such as companionship and responsibility. Motivation is a study in its own right which cannot be dealt with here; for further study the reader is referred to Maslow (1987).

In interface and dialogue design, attention has to be directed to important messages and actions which the user should take. Care has to be exercised that the design does not produce too many competing demands

for attention at once, thereby overloading the cognitive processor's ability to deal with events. Cognitive overload leads to malfunction and breakdown of the human machine, the symptoms of which are manifest in stress and task failure. While task overload is not the only cause of stress, it is an important facet of interface design. Stress can be caused by many factors such as worries about family life, social relationships, financial insecurity, etc. It is the interface designer's task not to add computer systems to this list.

2.6.2 Stress and Fatigue

Fatigue may result from continuous mental activity in over-long, monitoring tasks and from intense concentration in tasks demanding difficult mental activity. In either case, rest is required for the human mental system to re-adjust itself.

Fatigue can be caused by repetitive tasks containing no break points. Interface design should therefore ensure that long continuous tasks are broken up by rest periods in which the user is allowed to do a mental reset. These break points, called 'closure events', should be placed at natural intervals during operation of an interface. These intervals could be at the end of an operational sequence, such as entering a transaction record, or a search and replace operation in a word processor. The more complex a task, the more demanding and potentially fatiguing it may be. Break points should be planned with task complexity in mind, with more frequent break points provided to counter increased risk of fatigue.

Task complexity, however, does not always lead to increased fatigue. People find stimulating but demanding tasks interesting. Complexity may hold their attention and delay the onset of fatigue for some considerable time, although highly demanding continuous activity should be avoided because users may be unaware of their tiredness and make mistakes. Mundane, non-stimulating tasks are liable to cause user fatigue precisely because they do not stimulate interest and hence do not hold attention. Such tasks should best be avoided but if they are necessary, a high frequency of break points helps to combat the strain of enforced attention to an uninteresting task.

Fatigue can also be caused by sensory factors. Strong stimuli, such as bright colours, intense light and loud noise all cause sensory overload as they bombard the perceptual system and demand attention. If exposure to such stimuli continues for a long time, the cognitive system will try to ignore the steady state in the environment; however, such strong signals are not easily ignored. This sets up a conflict in the attentional process which can become fatiguing. Strong stimuli can also induce fatigue in receptors as strong light can cause eye and head aches, and loud noises may result in temporary deafness. Interface designers should avoid using too many strong stimuli.

2.7 Principles of Human–Computer Interaction

Having examined components of the human machine, its operation as a whole can now be described in a perspective of some of its limitations. Consideration of human properties of information processing allows a set of tentative principles to be drawn up, although care has to be taken in applying principles in practice, because the context of design has a strong effect on the validity of generalisations drawn from psychology.

Compared with computers, humans excel at heuristic, associative tasks but are poor with high volumes of data and repetitive tasks. People deal with complexity in the environment by imposing order on it and trying to automate solutions to problems. Classification, structuring of information, and skills are consequences of this propensity to organise and automate. Both humans and computers can process algorithmic and logical problems well, although computers produce much more reliable results. The great advantage people have over machines is a vastly more complex knowledge base even for things which we consider to be simple common sense and everyday knowledge. This coupled with the ability to increase that knowledge base by learning and reasoning heuristically gives humans an advantage over machine systems which will take a very long time to erode.

The human system is an associative reasoning machine. It deals with vast quantities of data from the environment by filtering it and abstracting interesting qualities from basic data. The system has to deal with multiple inputs, outputs and memory management as well as central processing (reasoning) according to a schedule which shares the limited resources of the system. The key point of designing for the human machine is to prevent overloading of its processing facilities, in particular short-term memory, and to harmonise design with human information processing. Hence principles which help memory and human reasoning abilities are important.

From knowledge of human psychology and the applied psychology of human performance it should be possible to draw up basic principles to guide the design of human–computer interfaces. Unfortunately psychology does not lend itself to such a venture as many explanations of human behaviour are still models and hypotheses, and in some areas little definite proof exists. However, some principles can be derived in spite of this limitation, although they have to be supplemented by justifications to substantiate them based on general utility, interpretation in a context, as well as empirical evidence.

Six basic principles are proposed:

Consistency: this is similarity of patterns which may be perceived in tasks, in presentation of information and other facets of an interface design. Consistency reduces the human learning load and increases recognition by presenting a familiar pattern. As we are pattern-

recognition machines, the more consistent patterns are, the less we have to learn, and the easier an interface will be to use.

Compatibility: between the user's expectation and the reality of an interface design. This principle follows on from consistency to state that new designs should be compatible with, and therefore based upon, the user's previous experience. If this is followed, once again recognition is enhanced, learning is reduced and the interface should be easier to use. The essential compatibility is between the user's mental model of the task and the task model embedded in the software by the designer.

Adaptability: interfaces should adapt to the user in several ways. The user should be in control, not the computer; so the interface adapts to the user's speed of work and does not enforce continuous attention. Also the interface should adapt to individual user characteristics, skill levels etc., as to do otherwise would offend the compatibility principle. Adaptability, however, must not be overdone otherwise the consistency of the interface is reduced.

Economy: this principle is based more on common sense than psychology. Interface designs should be economic in the sense that they achieve an operation in the minimum number of steps necessary to support the user and lessen the work of users whenever possible.

Guidance not control: interface designs should guide a user through a task with prompts and feedback information. The interface should function at the user's pace according to the user's command and should not attempt to control the user. This principle has two sub-components: predictability—users should be able to forecast what to do next from a system's current state; and reversibility—users should be able to back-track at will when mistakes are made.

Structure: interface designs should be structured to reduce complexity, because humans process information by classifying and structuring it within a framework of understanding. Structuring should be compatible with the user's organisation of knowledge and not overburden memory. This leads to a sub-component of simplicity and relevance; information should be organised so that only relevant information is presented to the user in a simple manner.

Principles are intended for overall guidance during design and as a set of criteria against which interfaces may be evaluated. To apply principles in the design process, they have to be translated into guidelines which pertain to different aspects of a human–computer interface. Guidelines, in turn, are modulated by the context of a particular application into design rules. Unfortunately, systems and people are complex; so to issue a simple set of guidelines for all situations may be appealing but in reality would only be misleading.

Designs need to be considered in terms of the objectives of creating good human-computer interfaces, which raises the question of assessment. The effectiveness of interface designs is frequently measured with terms such as usability, utility and efficiency. There are three basic concerns about the quality of an interface design:

- How well does it fulfil the users' objectives?
- How easy is it to learn and use?
- How much of it is used?

A design should aim to provide users with what they require in order to fulfil their objectives. This concept is common to systems analysis and interface design, that is, the matching of user requirements to the facilities provided in the system. In human factors terms this is called *task fit*—providing the appropriate tool to carry out a required task. A system may be easy to use and learn but if it does not do what the user wants it will be useless. Task fit is a consequence of the compatibility principle and mental models—the user's expectation of reality and what he gets.

Efficiency is often measured in terms of how easy an interface is to learn and use, combined with the inverse measure of how many mistakes are made. Generally it may be thought that there is a trade off between ease of use and ease of learning, but evidence points the other way; interfaces should be easy to learn and easy to use. Efficiency is a consequence of the economy, consistency and compatibility principles.

The concern for how much of an interface, and hence a system, is used is often ignored. The concept of usability aims to tackle this factor which may be caused by poor functionality in the task fit, by poor training, and by poor interface design. Users may be ignorant of or cannot be bothered to use a facility even though it may fulfil their task very well.

Compatibility relates to the concept of users' models, that is the users' mental model of how a system should appear and should work. This will be based on previous experience of computer and non-computer systems. It is the analyst's task to capture that knowledge and build the new system to be as compatible as possible with the users' expectations. Full compatibility may be technically impossible because of improvements to the logical design of the new system. Also user models differ on account of variations in individual experience; one single model cannot be completely compatible with each individual's view. The final design has to be a compromise with inter-individual variation.

2.8 Summary

Perception is the process of seeing and hearing. Images and sounds are received and coded in an abstract form as properties of the stimulus.

Interpretation is effected by comparing the input with long-term memory. Memory may supply a considerable amount of what we see and hear which creates illusions in some circumstances.

The human information processing machine is composed of sensory, cognitive, and motor processors with associated short-term and long-term memory. Short-term memory has limited capacity which may be expanded by increasing the level of abstraction of information. Information in short-term memory is held in chunk form and has to be refreshed frequently. Long-term memory has an infinite capacity and can be thought of as a highly networked database. Memory is essentially semantic, although two access paths probably exist, one via a semantic network, the other via abstract images.

Problem solving involves steps of formulating, searching and verifying problem solutions. A model of the process is a network of goal solution sub-steps, each of which has tests associated with it. The network is traversed by a strategy called a method. Various methods are used by humans, some of which are similar to search strategies employed in expert systems. Problem solutions are stored as skills and automatic processes which are called by a context. Mismatch of calling context and automatic behaviour can cause errors.

Human information processing is essentially sequential although considerable concurrent processing occurs. Sequential scheduling is controlled by attention which directs the resources of the cognitive processor. Attention has important consequences for task design. Fatigue affects attention and sensory processes and should be considered in task design.

From knowledge of psychology, six general principles of interface design can be drawn: consistency, compatibility, adaptability, economy, guidance and structure. These principles should increase the effectiveness of interface design which may be measured in terms of efficiency, task fit, and usability.

Further Reading

For general texts on cognition, Glass *et al.* (1979) or Lindsay and Norman (1977) give comprehensive coverage of the field. For more detail on perception Frisby (1979) gives a well illustrated description of vision, and Fry (1977) is a good general introduction to speech and hearing. A very readable account of memory, both working and long term, can be found in Baddeley (1979). For more advanced study, Christie and Gardiner (1987) contains chapters on most relevant topics in which the authors summarise research in their field and give guideline summaries. Card *et al.* (1983), besides being the source of the GOMS model, makes instructive reading, although it does view cognition in a narrow perspective.

3 Interface Analysis and Specification

This chapter covers the analysis phase of interface design in which information is gathered about users and the job they do. This work can be carried out and integrated with mainstream systems analysis. The steps involved are analysis of user characteristics, analysis of the user's job (called task analysis), recording user's perceptions and terminology relating to the system, followed by synthesising this information within the constraints of available hardware and system requirements to decide on the type of interface.

The steps involved in interface analysis are summarised in figure 3.1. One approach based on structured systems analysis techniques is described, followed by an overview of specification methods which have been developed within the field of human-computer interaction.

Tasks may be performed either completely by a computer, or completely by humans, or may be shared between man and machine. Computer tasks become part of the systems design, while human tasks become part of the human system job design. Shared tasks, however, require further analysis because they will impinge on both the human and computer systems.

Interface design takes an interface specification and reviews the system requirements in the light of human factors analysis. From this review a type of interface design is chosen and the main components of a design specified. The steps involved are task design to create the human system of work modules, job descriptions and operating procedures; design of the system support environment, that is, parts of the system to help the user; and design of the interface modules.

A brief survey of interface design types is given with the human factors properties of each type as a background to choosing design types.

3.1 Task Analysis

Task analysis is the decomposition of the activities within the system. It is a similar activity to requirements analysis as practised in systems analysis and design, with the added proviso that in task analysis all the system tasks, including human-related actions, are described and not just the functions to be computerised.

Methods of task analysis in human-computer interaction have not been well defined; in view of this and the close relationship of task analysis to systems analysis, it is appropriate to employ a method borrowed from the latter discipline. A good method for this purpose is the techniques of Structured Analysis as described by De Marco (1978) and Gane and Sarson (1979). The method decomposes the system down into smaller units called *functions*; these are discrete pieces of work which achieve one goal. Functions may be directly equated with tasks. Each function takes data in, does something to it (transforms the data) and then passes it to the next function. When a series of functions are linked together they form a description of how the system operates in terms of its components. By linking functions with the data connections, called *dataflows*, a map of the functions within a system can be built up as can be seen in figure 3.2. This approach, called *functional decomposition* by systems analysts, is the essence of task analysis.

Functional decomposition segments systems into smaller units each of which is identified by a goal or purpose—in other words the function which the system component carries out. By successive refinement during analysis an increasingly detailed view of the system is obtained, first at the sub-system level, then at sub-sub-system level and so on. When the units are reasonably small their contents can be described as actions in a procedural sequence. The procedure consists of the necessary steps for carrying out the task. Tasks are composed of groups of actions which achieve a purpose; combined together they either form the part of a person's job within the system or, possibly, an automated activity.

Actions, described by verbs, are the primitive building blocks of tasks which cannot be decomposed further without losing meaning. For example, in an order-processing task, actions could be Check-Customer-Credit-Limit, Calculate-Order-Lead-Time, Determine-Order-Discount. However, although Check-Invoice-Payment, Allocate-Stock-to-Order are actions, Check-Days-not >31 in Estimate-Order-Date is probably too low level to be an action in its own right. Although the level of decomposition is a matter for the analyst's judgement, further decomposition would specify the logical operators of the comparison or mathematics for a calculation, and this renders the description as a whole meaningless.

When to stop subdividing functions is a matter of judgement and experience, but one heuristic method is to subdivide until each function achieves a single purpose and procedural detail of how it works can be described in roughly half a page of concise English (for example, 6 to 12 steps).

Connections between functions are by data flows. Other diagram components are data stores or files (open-ended boxes) and external entities (squares) which supply data to or receive data from the system. The whole task structure of the system can be illustrated using data-flow

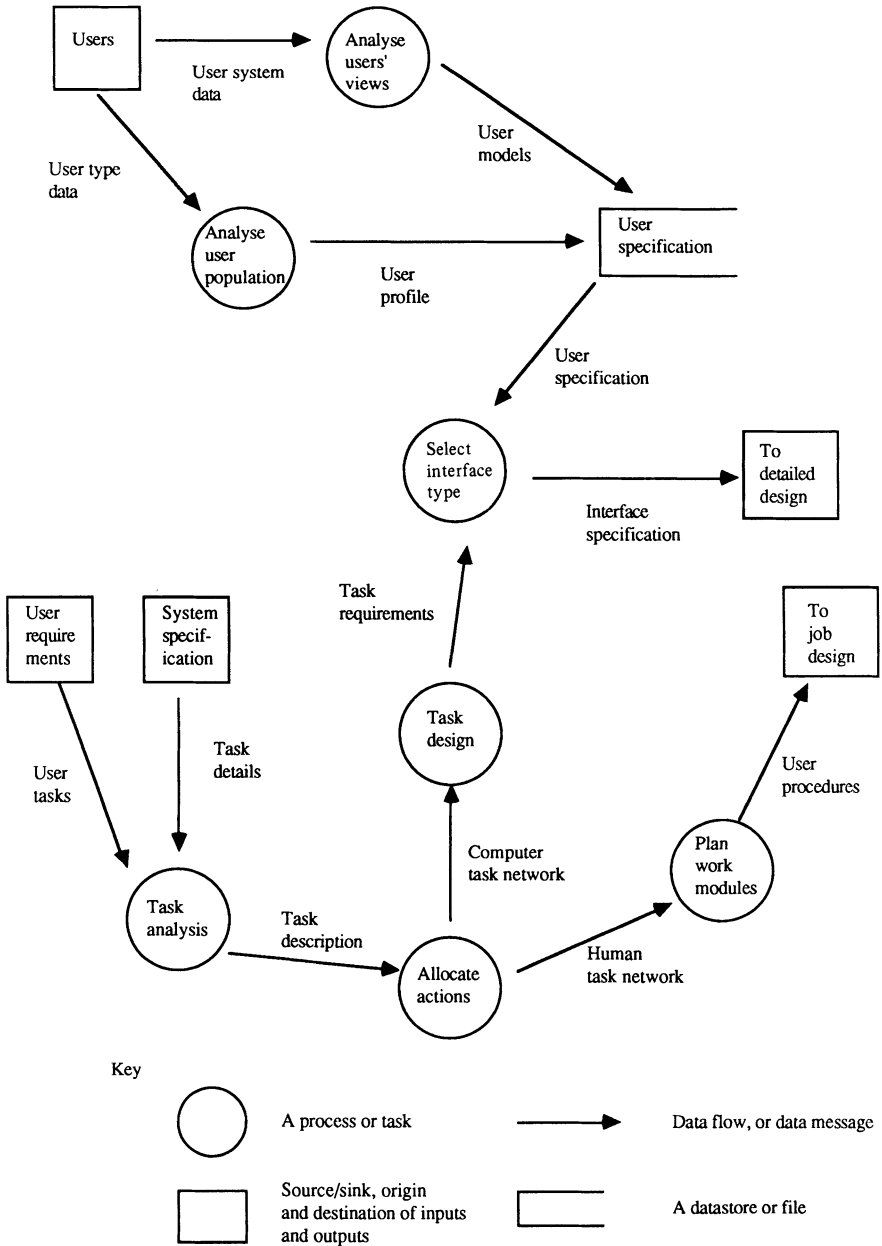


Figure 3.1 Flow diagram showing steps in interface design.

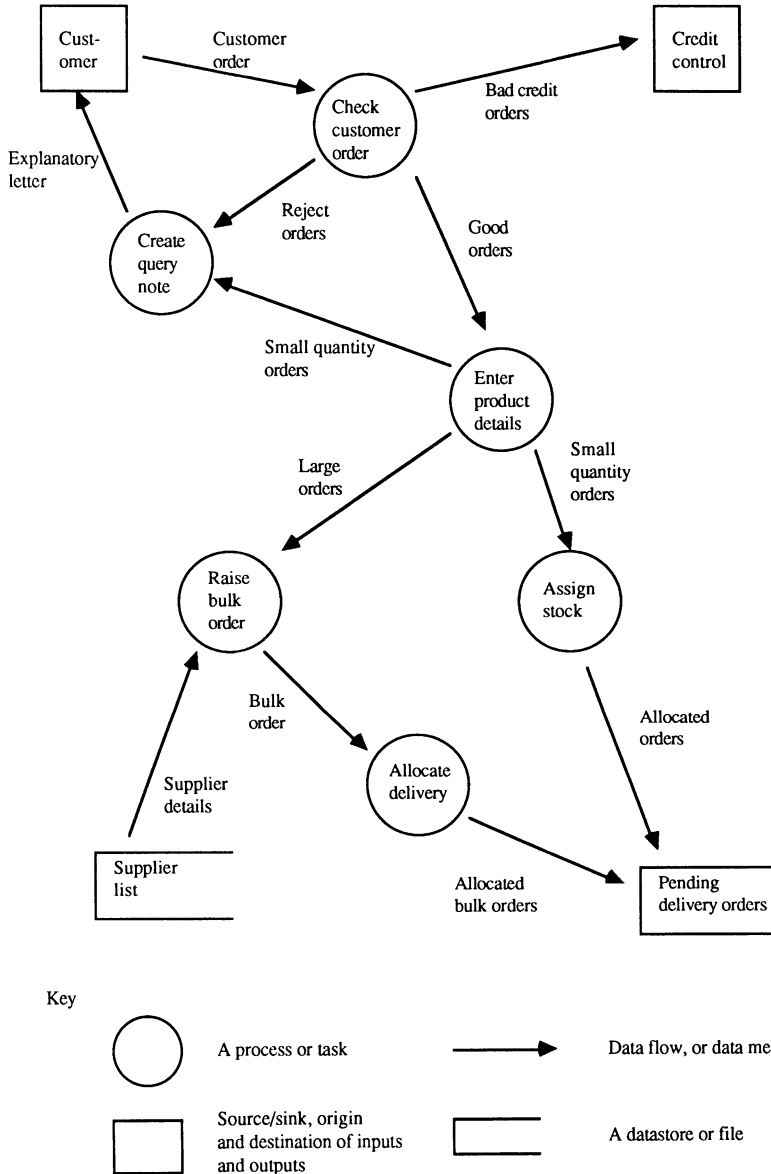


Figure 3.2 Data-flow diagram of an order entry task. Orders are checked against the customer's credit rating and then split according to the size of the order. Small orders are satisfied from stock while large orders are sub-contracted to other suppliers.

diagrams to show a map of sub-tasks communicating by data messages which are passed between them. Further description of functional analysis and data-flow diagramming techniques can be found in De Marco (1978) or Gane and Sarson (1979).

Using structured analysis techniques, task detail can be described in narrative or more formally as a sequence of actions in Structured English. Structured English is a constrained sub-set of English composed of a set of reserved words for expressing sequence control (If, Then, Else, Repeat, etc.), verbs which describe actions, nouns to describe data, and conjunctions (see figure 3.3). It describes the procedural detail of how a task is carried out in terms of sequences of actions, alternatives and repetitions. The reserved word set combined with indentation of the text show the scope of control.

Structured English describes the sequence in which the actions are performed and any exceptions to that sequence. Once a task has been described the next step is to allocate all or parts of it to either man or machine. Task and action allocation is the first step of task design which is dealt with in the section 3.4.1.

3.2 Analysing User Characteristics

Human-computer interfaces should be built to suit the needs of people, consequently it is important to discover what types of people will be using

```

Task: Loan-Books

Repeat WHILE Borrowers
  Request reader-ID
  Check reader-ID
  IF Reader-ID Invalid
    Pass to Membership-Check
  ELSE
    Continue
  END-IF
Repeat WHILE book-requests
  Enter book-mark on borrower-slip
  Write reader-ID on borrower-slip
  Stamp book with return-date
  Remove book-in-library-tag
END WHILE
END-WHILE.

```

Note the reserved words are in capitals, actions are verbs with the first letter in upper case and data items are in lower case with hyphens to make the name continuous. Indentation is used to show the scope of control for IF-THEN-ELSE constructs, etc.

Figure 3.3 Structured English for book loans within a library system.

the interface. Groups of users vary in their knowledge of computers, general abilities and in a variety of factors which affect their ability to deal with an interface. Therefore, the objective of user analysis is to obtain a thorough knowledge of the skills and experience of all users in order to be able to predict how they will react to different interface designs. This enables sound judgements to be made when matching the sophistication of the interface to users' abilities.

3.2.1 User Categories

Users have been categorised by many authors in a variety of schemas intended to describe user classes which have important implications for interface design. Four main categories of user are generally distinguished:

Naive: users who have not previously encountered computer systems. They may show fear of computers, will be unfamiliar with their operation, and will have little or no knowledge about the system. Completely naive users are becoming rarer as computerisation spreads, but this user class will still be encountered when introducing computers into a non-automated environment.

Novice: users with some experience of computers, although they may be unfamiliar with a new system. They will probably have little knowledge or experience with the system and are liable to make many mistakes, consequently they need considerable support. Most users of new systems start as novices and progress with experience to becoming skilled; although if usage frequency varies they may regress to novice status after a period of inactivity.

Skilled: users who have gained considerable experience with a system and are proficient operators. Most frequent users become skilled with time and require more economic, rapid-to-use interfaces with less support than novices. Skilled users, however, do not have much knowledge of the system structure so they are unable to repair unexpected errors or extend the system capabilities. Instead they are skilled at operating one or more system tasks.

Expert: experts are distinguished from skilled users by their knowledge of the internal system structure. Experts generally have some computer software expertise, good knowledge of how the system operates and an ability to maintain and modify the basic system. Experts need a sophisticated interface which provides facilities that enable them to modify and extend the capabilities of the system.

Although the above categories provide a workable framework for analysis, user classification is rarely so simple. Within a user population there may be a mix of people who have used the system for a long time,

that is, skilled users, and new recruits who will be novices. Variation occurs even within individuals over time. Expert users may rotate jobs and not use a system for several months, during which time they will forget their knowledge and may regress to a novice state. In spite of these difficulties, measuring user characteristics is worth while because it enables the designer to choose an interface type and level of support which is appropriate to most of the users.

3.2.2 *Measuring User Characteristics*

To start classifying users, some basic metrics are required. These are a mixture of anticipated usage patterns and observed abilities of the user population. The critical factors are how often people use a system, how much they already know about the system and how much they may be prepared to learn.

The choice of these measures is linked to expected user performance when operating the system. For instance, frequency of use will affect how skilled users become, computer familiarity will indicate how much training may be necessary to attain skills; system knowledge and experience with computer software may be used to predict how much knowledge users may acquire and their expectations of the sophistication of an interface, and discretionary users are usually less tolerant of poor interfaces than users who have no choice about using a system. The important measures are:

- *Frequency of use*: how often will the system be used? Frequent users build up skills and become experienced quickly; if use is infrequent then skill build up will be slower and a more supportive interface may be necessary. The variation in usage frequency over time is also important. If frequent users have gaps between using the system, then they may forget key information and require help facilities
- *Discretionary usage*: use of a system may either be compulsory, that is, part of someone's job, or it may be an optional extra, for instance, a data entry clerk may have to use the sales-order-processing system as part of the job duties, but it is up to a manager whether or not to use VisiCalc for forecasting. All interface designs should be good, but interfaces for discretionary users have to excel in ease of use and attractiveness to users, otherwise the system may never be used
- *Computer familiarity*: most users have some experience of computers but the degree of experience varies. This measure will have important implications for user training
- *User knowledge*: some users may have considerable knowledge of computer programming and operation. These expert users have the ability to extend the functions of a system and its interface; consequently they will need a flexible programming or command language type of interface to satisfy their aims

- *User mental abilities*: this is a measure of the general knowledge and intelligence of users. It is necessary to judge the level of interface sophistication which users can deal with and how much they may be expected to learn about an interface
- *User physical abilities and skills*: the physical characteristics of user populations and workplace design properly belong to the realm of ergonomics. Information should be gathered at this stage especially if new equipment and the workplace environment are being designed. The objective of ergonomic analysis is to choose equipment which is designed to meet human needs; however, such considerations are beyond the scope of this book and the reader should consult Damodaran *et al.* (1980) for further details. The relevant skills within the context of design of interface software are experience of any interface-related skills such as typing, use of a mouse, etc.

Using these measures, user populations may be scored on a simple scale (such as 1 to 10 where 1 = low frequency, 10 = high frequency). It is important not only to establish a picture of the average characteristics of the population but also of the variation, as the interface will have to try to satisfy different types of users. This information forms part of the interface-human requirements specification which feeds through into the strategic choice of interface type.

In the example shown in figure 3.4, librarians were expected to use the system as part of their everyday job; therefore, their usage frequency was high; however, some of the librarians rotated jobs, and hence they had a high range in frequency of use. Few librarians were familiar with computers and likewise few had knowledge of automated library systems or computer systems in general; most, however, were of above average intelligence. The metric derived from this analysis points towards a supportive dialogue which is not too sophisticated. The measures of user characteristics are used in the selection of the dialogue type which is appropriate for the user population.

3.3 User Models and Views

User models come in several varieties depending on the interest of the authors. The terminology is further confused by ambiguity about who constructs the model, and what is being modelled. User models can be inside the user's head (often called *mental models*), the designer's idea of what is inside the user's head (*conceptual models*), and finally a piece of software enshrining the designer's model. In this section we are concerned with acquiring the user's mental model of a system in the form of a designer's model which is used to help construct the interface. To put user

System ID : Automated Loan system
 Population ID : Assistant Librarians

	Median	Range
Physical skills	typing (some)	
Discretionary	No	
Frequency	8	2-10
Computer Familiarity	2	1-4
User Knowledge	1	1-3
Mental Abilities	6	4-8
Population score	17	8-25

This score can be summarised for the two most important variables using the following table:

	Total score
Knowledge	
Abilities	20 - 15 - 10 - 5 - 0
Sophistication	+++ ++ + -
Frequency	
Familiarity	20 - 15 - 10 - 5 - 0
Support	- + ++ +++

In this case the librarians rate low on the sophistication scale (7 out of 20) and average on the support measure (10 out of 20)

Figure 3.4 Sample analysis of a user population.

models in perspective, the following types are current in the human-computer interface literature:

- *Theoretical cognitive models* constructed by psychologists in order to understand human mental processes. Information processing models, as used in chapter 2, fall into this category
- *Models of user knowledge*. These models are inspired by CBT (Computer Based Training) interests and adaptive interfaces. The model attempts to capture the knowledge categories in a domain and the inter-relationships between the categories. Models can then be constructed of each user's knowledge to assess how users learn by traversing the knowledge network. In adaptive interfaces the model attempts to describe the user's knowledge in terms of plans and procedures (see chapter 10 for more details). These models are embedded in software
- *Models of user characteristics*. These models attempt to classify users in broad terms of skill and ability, as described in section 3.2. They are also called user profiles
- *User task models*. The user's concept of how a task is constructed in terms of its functions and operational sequence

- *User views.* The user's models of the system structure which may be expressed either in terms of visual metaphors (for example, an office and its components) or in a verbal classification of system components. They are also called the user-system image

In this section user task models, user views and models of user characteristics are examined; theoretical models and models of user knowledge are not discussed because they have less direct relevance to the standard practice of interface design. User views are the way in which users describe and visualise the structure of the current system. User task models are an attempt to discover how much users know about the system in terms of its operation and what are their expectations about how it will work. The importance of user models lies in the compatibility principle: the more an interface conforms to users' pre-conceived notions of how it should appear and operate, the easier it will be to learn.

Most users construct mental models of systems based on their past experience of the system and similar computer systems. Experienced users are more likely than novices to have well-formed models. When the system is first encountered the user's model may be vague but it will grow as experience increases. It is the interface designer's responsibility to make the interface conform as far as possible to the user's previous model and, if no previous model existed, to present a clear structure of the new system and make assimilation of the new system model as easy as possible.

User models are discovered during task analysis and may be found in the names people use for objects and functions in the system, the connections they make between tasks, and the visual and verbal metaphors they use to describe the system. User models may have varying degrees of accuracy; for instance, there is the designer's model of what he thinks the user expects, and the user's model of what he expects of the system. It is the analyst's job to make sure that these models coincide.

Task models and user views have two dimensions: first, a static view of the system structure in terms of objects and their relationships which may be expressed in visual or linguistic terms. This will contribute towards design of interface presentation. Second, there is an expectation of system operations, or the dynamic behaviour of objects which is relevant to the dialogue design. The models are made up of a structure, either in terms of static objects and relationships, or procedural sequences of activity and a set of descriptive labels by which users identify objects and operations within the system. For example, in banks, dealers see foreign exchange dealing not as money but as 'deals' which come in a variety of types, 'spot', 'forward' and 'overnight'; deals are not placed but 'struck'. The descriptive labels form the semantic, language-based view of the system. In addition users may have spatial metaphors for their system, expressing the physical

system layout and possibly a view of abstract objects described in visual-spatial terms.

In a library, for instance, users may view the system as books which reside on shelves which are organised in stacks which, in turn, occur in subject areas; all of which may be visualised in terms of a hierarchy. There may be other parts of the library which serve specific purposes, such as a reference section, temporary stacks, books to be reshelfed and the issue desk, which are seen in spatial terms as a network. Such system image views can contain a rich description by which users organise their knowledge about the current system in terms of a physical layout or map (see figure 3.5).

User view analysis is important for presentation of the interface because the users' terminology should be employed if possible. Furthermore, visual metaphors of the system may be directly transferred into an iconic form as demonstrated by the Xerox office-desktop layout in the Star workstation. User views can also help in dialogue design because the view can also reflect the functional organisation of systems components from the user's angle. This can suggest ways in which components of the interface are put together to ensure that the sequence of operations in the new system matches the old.

3.4 Task and Job Design

The task specification resulting from analysis may not be well organised or even achieve the user's objectives. Task design aims to re-organise the task specification to produce a more logical organisation. Tasks are then grouped into units of work which will become a job description. This involves synthesising tasks with differing characteristics into a job suitable for people, and planning the work so that the workload is matched to the personnel resources available.

Tasks vary in complexity in physical and mental dimensions. Design aims to create human tasks which are neither too demanding (that is, composed completely of very complex steps) nor too simple which may lead to the operator becoming bored. Variety is desirable in any task. Task complexity also has to be matched to personnel ability, hence the capabilities of the user have to be considered when designing tasks. It is no use giving someone a stimulating yet over-demanding task in relation to his or her abilities. A compromise has to be reached which ideally should give people tasks which stretch their abilities thereby encouraging them to develop new skills and widen their experience, while not going beyond their abilities, because that would cause despair and frustration.

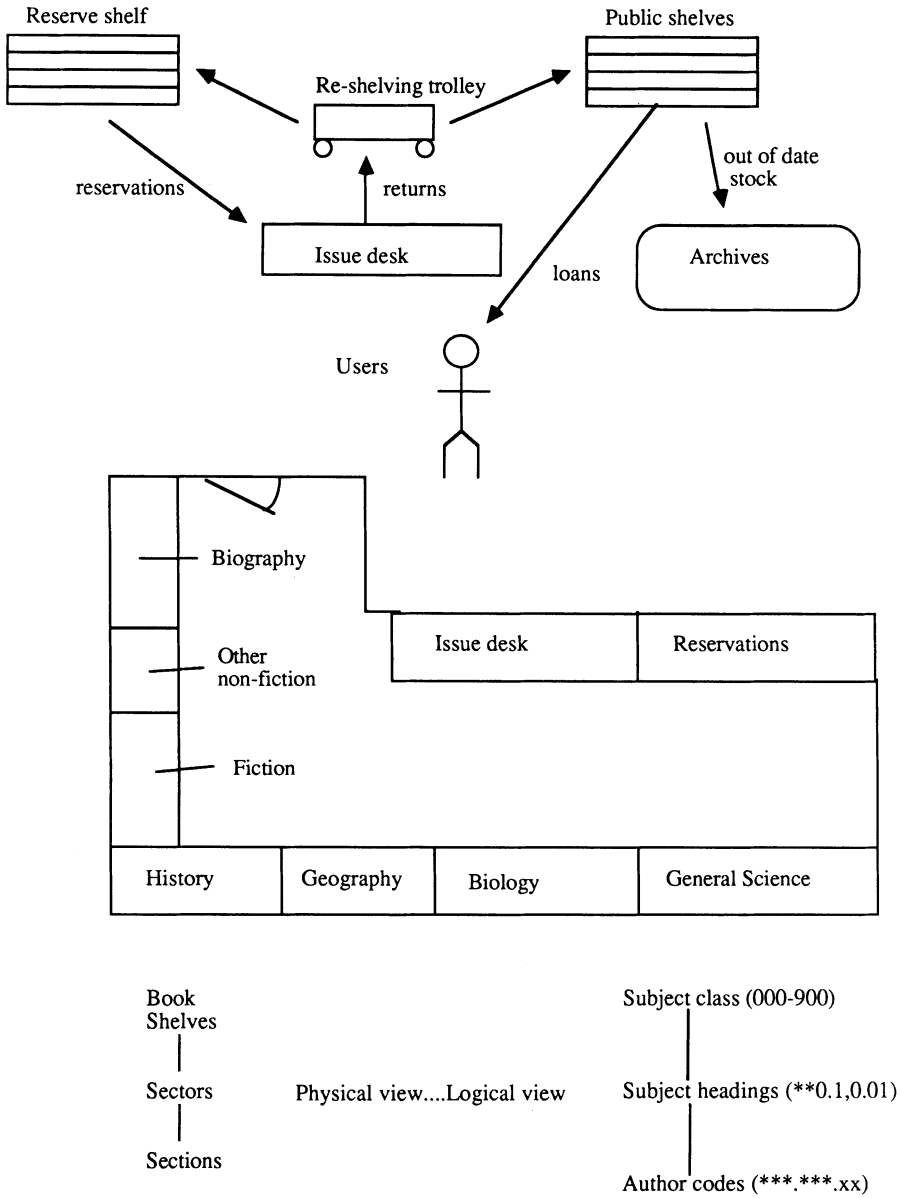


Figure 3.5 User view in terms of a conceptual model describing a library in physical terms of layout and book movements and in logical terms of book classification.

3.4.1 Task Allocation

Within each task, actions are allocated to either the computer or the users, or to both users and the computer. Generally users should receive tasks which require initiative, judgement and heuristic reasoning. On the other hand, computers should get repetitive checking, calculations and data-handling tasks. Data entry, data retrieval and decision support are examples of mixed tasks in which human and computer interact to achieve an objective. Mixed tasks require further refinement to specify the human and computer components.

Allocation produces two task networks: one human task network and one for the computer system. Both networks can be designed using data-flow diagrams to illustrate the logical sequences of tasks. The human network will form the basis of operating procedures and the user manual, and the computer network will add to the system specification. Task allocation could be done on the data-flow diagram at the level of a process/task but this level obscures much of the interface operational detail. Hence it is preferable to allocate parts of the system to either human or computer at the action level which is shown in Structured English. The steps are:

- Inspect the data-flow diagram and mark tasks as either for the computer system, human system or joint tasks
- Take the joint tasks and allocate actions within each task to either human or computer
- Construct new computer system and human system task networks

Cooperative actions involving both the user and computer require further refinement to specify how the human and computer are to interact. In the task sequence, illustrated in figure 3.6, most of the repetitive checking is given to the computer with the human operator supplying the input; however, in the Raise-Bulk-Order function there is a complex step which involves a trade-off between delivery date and sourcing the stock from one or more suppliers. This decision involves local knowledge of delivery dates and heuristic judgement to find the best delivery date in relation to the number of suppliers trade-off, and consequently this task is allocated to a human. It may be possible to computerise this function with a small expert system, but the designer should preserve human interest and activity in a system so this step is left uncomputerised.

The design elaborates separate human and computer actions in terms of human actions and computer support for those actions, such as displaying information, suggesting options, giving warnings etc. For instance, the computer may be required to provide decision support for the human operator. In the above example, finding the best suppliers involves browsing down a list with different combinations of product categories and suppliers. How the interaction for decision support will operate has to be designed and agreed in consultation with the user.

H: = allocated to human operator
C: = allocated to computer

Function: Check Customer Order

Repeat while orders

H: Enter customer number
C: Check customer number
C: If no number assume customer & Pass to accounts
C: Check customer against credit-control-list
C: If customer on list Send to credit control
H: Check order value against customer order credit limit
C: If over limit Send to credit control

Function: Enter Product Details

Repeat for products ordered

H: Enter product code
C: Check product code
C: If in stock Tick ex-stock column
C: If not a stock item Tick direct column
C: Check product quantity
C: If less than minimum quantity Raise query note
H: If over delivery limit and not a stock item
 Raise bulk order

Function: Raise Bulk Order

Repeat for High quantity products ordered

H: Enter stock category against product code
H: Create list of suppliers who have appropriate stock categories
H&C: Find minimal number of suppliers who can deliver all the categories ordered
H: Find suppliers who have quickest delivery dates for ordered categories
C: Write out bulk order to suppliers

Function: Calculate Delivery Details

H: Enter estimated delivery date
C: Check delivery details present
C: If absent Raise query note
H: Stamp to authorise order

Figure 3.6 Task description in an order entry system.

Tasks within a system may not always fall into a well defined network. Some tasks naturally occur in sequences; in other systems task operation is fragmented and each task may be performed independently. Processing a sales order or a library book loan are examples of structured task sequences composed of a series of sub-tasks having to be carried out to achieve the goal. On the other hand, many office tasks are unstructured; a manager may write a memo, book a meeting, answer the phone or analyse sales figures in an unpredictable sequence. In this case there will be no human task network, just a set of unrelated tasks which the user needs to

access individually. The structure of task sequences is relevant to how tasks are accessed by the human–computer interface.

A further consideration in task sequence design is to provide break or rest points within the task. A continuous sequence of activity causes physical and mental strain which can lead to loss of concentration and errors. Careful design of tasks with break points at regular intervals allows concentration to be refreshed by a closure event; this is a short period during which the cognitive processor can be reset. Closure events should be planned at logical end-points in a sequence, such as the end of a record, after each query, etc. If there are no natural break points in a sequence, closure events will have to be imposed on long task sequences at intervals of 2–5 minutes depending on complexity of the activity involved.

Finally the mental load on the user should be estimated. The objective here is to reduce overloading on short-term memory; so at each task step the quantity of information required by the user should be calculated. The design is then checked to establish if the information is readily available to the user in a display or if it has to be held in short-term memory. Memory loading is particularly important at decision points and error recovery within tasks, and care should be taken that users do not have to hold too many facts in working memory. Also users should have all the information available for the action they are engaged in, and not have to remember data displayed in previous task steps.

3.4.2 Work Module and Job Design

Task design in its fullest sense involves job design, which aims to match task demands to the operator's abilities and to provide jobs which give people the correct amount of interest, responsibility and satisfaction. To treat such matters in detail is beyond the scope of this book; so the aim of task design within this limited context is to provide a better understanding of the designer's problem when designing the human part of an interface. The human side of the interface forms the basis of system operation manuals, training documentation and user guides.

Tasks should be measured on a simple scale to establish their human factors properties:

- Complexity—in terms of reasoning, judgements and decision making
- Concentration—attention to detail, and the monitoring activity necessary to complete the task successfully
- Responsibility—importance of task in overall system; consequences of task failure
- Variety—variability of task in one of the above measures

A small number of tasks are combined into one work module. A work module is an identifiable piece of work which will be performed by one person to fulfil one system objective. An illustration is the tasks involved in

order entry: data input, customer credit clearance, resolution of errors and credit queries. Another work module could be order progress chasing: determining where orders are in the system, identifying key late orders, investigating reasons for delays and proposing solutions.

Work modules should be balanced in terms of complexity and concentration. Too many repetitive undemanding tasks will cause attention to wander; on the other hand, too many demanding tasks will cause fatigue. The correct balance should provide stimulation and interest without fatigue, ideally by a mixture of undemanding routines mixed with more challenging decision making. Task flow within modules should be examined to make sure task overload does not happen. Overload is caused by too many things happening at the same time. Many tasks may require the user's attention simultaneously, swamping the user's capacity with conflicting and urgent demands; as a result nothing gets done, leading to task failure.

Task overload may not be apparent within normal operating procedures even if they are well planned; instead it occurs when errors or the unexpected happen. If the demands of error processing are poorly or incompletely specified, task overload may be the result. Expected frequencies of errors should be calculated and work time allocated to the resolution of such errors.

Another common cause of task overloading is peaks in workload. For example, in many transaction-processing systems inputs come in bursts, such as telephone orders at the end of a day or a peak of mail orders in the morning. Calculations should be made for the time it will take to process input at peak loading as well as at average input rates. Manpower has to be allocated to deal with the load within the constraints of cost because peak rates at one part of the day usually imply low rates at other times. It is uneconomic to have staff employed for processing the peak load completely unoccupied at other times. Part of the task designer's job is to plan the workload so it is as even as possible, allowing time for error processing. Manpower is then allocated to carry out the planned work, matching the skill levels of individuals to the demands of the work.

3.5 System Environment and Support

Interfaces do not exist in isolation. The interface functions in an environment which influences the performance of the interface and may impose constraints upon it. Design of the interface/task environment has two considerations:

- *Physical design of the workplace.* This subject is within the realm of ergonomics and readers are referred to Shackel (1974) for more details

- *Design of interface support documentation.* This consists of the user manuals, technical documentation, training courses and training manuals

Design of user documentation

User manuals can be based upon the human part of the task specification. Two types of documentation are produced for most systems:

- *User operations manual*—this gives instructions on how to use the system
- *System technical documentation*—this is intended to explain the structure and internal workings of the system and may be produced at different levels of complexity for system support programmers and skilled ‘local expert’ users

User manuals should be clear, concise and well structured. It is a well known complaint that users never read the manuals, a symptom usually of poor manual design. Users have two broad requirements of documentation:

- *Education*—to find out about the system and how to operate it in its early stages of implementation
- *Aide memoire*—to access a specific piece of information quickly and often in an emergency

These demands conflict. The first requires a well structured guide which leads the user systematically through the system, while the second is for direct access to a specific point. Add to this people’s propensity not to read massive amounts of documentation and the problem becomes apparent.

It may be solved by writing three separate documents:

- The training guide which introduces the user to the system, aimed basically at education
- The quick guide for users who are too lazy to read the whole training guide and need only the bare minimum of information to start using the system
- The reference guide for trouble shooting and aide memoire later on

The training guide should be well structured to lead the user through various facets of the system one at a time, allowing one area of knowledge to be acquired before moving on to the next. Quick guides should contain commonly used command sequences with minimal instructions for operation and exhortations to read the training manual if the user gets into trouble. Reference guides should be laid out in an itemised manner with indexes and clear access paths to data. More general guidelines to help user assimilation of information, which can be applied to all guides, are:

- Structure information in a hierarchical manner: chapters, sections, paragraphs, etc.

- Label sub-divisions with clear headings and codes to show the relationship: 1, 1.2, 1.2.1, etc. Indentation can be used to further clarify hierarchical classification
- Paragraphs and sentences should be short and to the point
- Instructions and text should be jargon free, unless the user's own terminology is being used
- Procedures should be laid out sequentially and numbered to show the steps
- Important steps should be highlighted using bold characters, different fonts, colour or icons
- Use pictures, diagrams and visual methods to illustrate points if possible
- Keywords should be placed in the margin to provide direct access to specific topics
- Point by point summaries should be given at the end of chapters

Many of the above points are illustrated in figure 3.7 which shows part of a well designed user manual.

3.6 Interface Design Styles

Having designed the tasks the next step is to decide the interface design styles to be used. This is a trade-off decision, intended to match the users' characteristics with a suitable design style within the constraints of the system tasks and available hardware and software.

Several different types of design have been created for human–computer interfaces. Each type has different qualities and capabilities; consequently, when choosing the correct interface type or types for a particular set of system and user requirements, designers have to be aware of the merits and limitations of each particular type. This section surveys interface types and their characteristics. The important criteria for judging a match of an interface design to a user population relate to the qualities of the dialogue style, that is, how many functions it can provide, how sophisticated it is, and how easy or difficult it is to use. The capabilities and ease of use are matched with system requirements before considering the effort and cost of interface development and then making a final choice. Most interfaces use more than one design style, each style being matched to the requirements of a task or group of tasks, while the overall design aims to provide the correct level of sophistication and support for the user population. The criteria which will be employed in the survey are:




- *Ease of use*: how easy to use is the interface for inexperienced users?
- *Ease of learning*: how easy are the interface commands and functions to learn?

DEL imm & def

DEL linenum1 , linenum2

DEL deletes the range of lines from linenum1 to linenum2, inclusive. If linenum1 is not an existing program line number, the next greater line number in the program is used in lieu of linenum1; if linenum2 is not an existing program line number, the next smaller program line number is used.

If you don't follow the usual format, DEL's performance varies as indicated below:

<u>syntax</u>	<u>result</u>
DEL	?SYNTAX ERROR
DEL ,	?SYNTAX ERROR
DEL ,b	?SYNTAX ERROR
DEL -a[,b]	?SYNTAX ERROR
DEL Ø,b	deletes line zero, regardless of the value of b.
DEL 1,-b	ignored, even if the program's smallest line number is zero.
DEL a,-b	?SYNTAX ERROR if a is greater than the program's smallest line number, unless the program's smallest line number is zero and a is one.
DEL a,-b	ignored if a is not zero and the only program line is line number zero.
 DEL a,-b	ignored if a is not zero and if a is less than or equal to the program's smallest line number.
 DEL a[,]	ignored.
 DEL a,b	ignored if a is not zero and a is greater than b.



When used in deferred execution, DEL works as described above, then halts execution. CONT will not work in this situation.

Figure 3.7 BASIC Programmer's User Manual for Apple II Microcomputer. Note that the eye symbol is used to draw attention to important features; the layout gives a clear link between cause and effect, and capitals for emphasis.

- *Speed of operation*: how efficient is the interface in terms of operational steps, keystrokes and response time to achieve a particular operation?
- *Sophistication*: what range of functions are provided and can functions be combined in new ways to extend the power of the interface?

- *Control*: does the user or the computer initiate and control the dialogue?
- *Ease of development*: how easy is the interface to design and how much development effort will be required?

The dialogue types are ordered by approximate complexity in this overview. A more detailed analysis of dialogues is contained in later chapters.

(a) *Question and answer*

This is a simple type of human-computer dialogue which is a sequence of questions (or computer prompts) followed by answers (human replies), as illustrated in figure 3.8. The human replies are usually restricted to a Yes or No (Y/N) in the simplest case; more complex versions allow numeric and alphanumeric code replies. Replies are invariably restricted to a small set of valid responses, therefore the sophistication of this dialogue type is severely limited. Question and answer dialogues are easy to use and learn because the prompts should give complete instructions to inform the user what to do by listing the valid responses. This interface type is also easy to program as replies can be validated by simple conditional statements or against a small look-up table.

Advantages

Easy to use
Easy to learn
Easy to program

Disadvantages

Unsophisticated
Slow to use

Suitable for: naive and novice users, with simple conversational systems. Computer-initiated and controlled dialogue.

(b) *Menus*

A menu is a simple dialogue type suitable for inexperienced users. All the choices available are displayed as prompts on the screen; the user selects one, usually by a single character or digit code, the code number or letter being displayed beside the option description (see figure 3.9). Menus are limited in the number of choices which can be displayed on a screen at one time. Ideally there should be up to nine choices; more than this overloads short-term memory and increases the search time within menus. As a result, systems with many options have menus organised in hierarchies to provide a logical access path. This is simple to use for inexperienced users but slow and tedious for expert users who have to page through many menu levels to access the option that they want.

Menus are used most frequently as an access mechanism; however, they can also be used for data entry when there is a choice between a limited

Patient Administration System

Enter Patient Code (or E to exit): >220345

Margaret Smith: admitted 12/12/87

Enter selection for patient history

D for diagnosis
T for treatment
X for X ray results
L for laboratory tests
C for consultation list

>L

There are no laboratory reports for this patient

Do you want another option for this patient ? (Y/N)>N

Do you want to access another patient history ?(Y/N)>N

Patient Administration System

Enter N to add a new patient

O to change or delete an existing patient record
E to exit

>

Figure 3.8 Example of a Question and Answer Dialogue. The user is prompted for either a simple Y/N answer or a small valid reply set in a 'mini menu'. The conversation proceeds in a series of short question and answer steps.

number of items. Once again the limitation of menus is the number of items which can be displayed on one screen.

Menus are simple to program and easy to make 'bullet proof' for the users; that is, all possible invalid responses a user can make are trapped by the program and an appropriate error message is displayed. The user can select valid choices, escape and possibly a help option, but all other keys invoke error responses. For the information they provide to the users,

Local Information Database

Top level

1. News
2. Weather
3. Sports
4. Travel
5. Shopping
6. Markets
7. Education - courses
8. Libraries - reference services
9. Local authority services
0. Finish

Enter your choice:

(Type ? for help on how to use the database)

Figure 3.9 An example of a menu dialogue. Options are chosen by entering the number alongside the desired subject category.

menus are resource hungry because a whole VDU screen has to be transmitted for every menu. Although some types of intelligent terminals can minimise the transmission load, transmission of whole screens of information for one reply can be a significant penalty in response times for systems using remote terminals.

Advantages

Easy to use
 Easy to learn
 Easy to program

Disadvantages

Slow to use in large systems
 Limited choice per menu
 Transmission overhead

Suitable for: novice users, with interfaces designed to access system options. Computer-initiated dialogue.

(c) Icons

Pictures or icons are used to represent functions on a menu-like display. Figure 3.10 illustrates the use of icons for a personal computer interface. To select a function, users point at an icon with the cursor employing a mouse pointing device. Icons are a very effective technique if the icon-pictures are realistic, because the learning time is reduced and operation becomes very easy for inexperienced and experienced users. This technique has been used extensively by Xerox and Apple and has the potential to generate an international dialogue language which transcends language barriers.

Icons, however, have limitations of individual differences in interpretation and therefore usually have to have some clarifying text associated with the image. Also icons take up a considerable amount of space on VDU screens so the technique is no more economical than standard menus when displaying a large number of choices. Icons create meaning by being realistic, which works well for concrete objects such as files (a filing

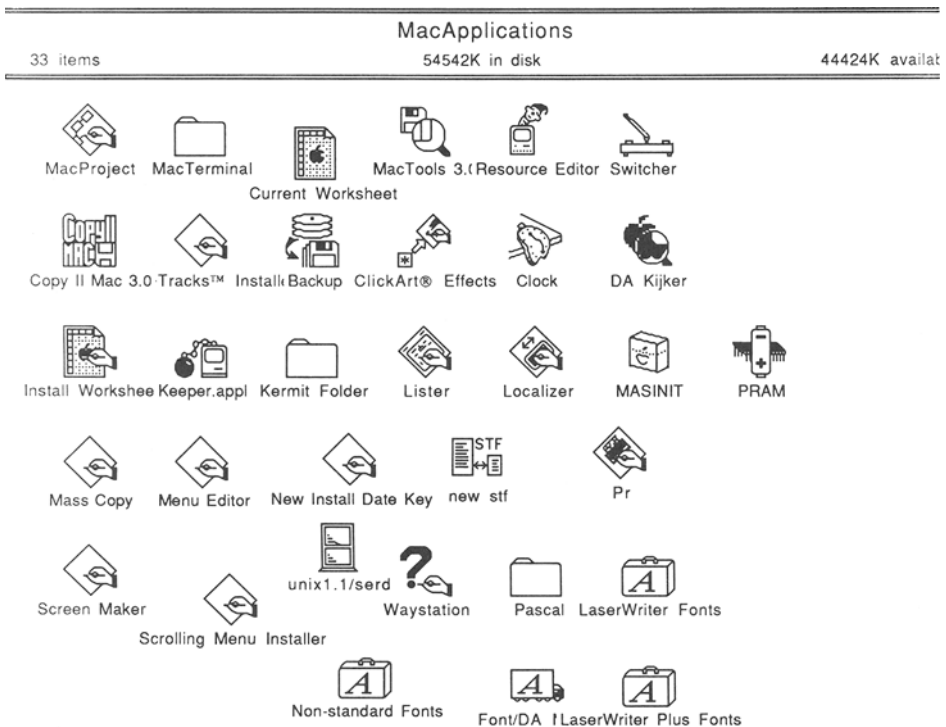


Figure 3.10 Use of icons to illustrate system facilities.

cabinet), messages incoming (an in-tray full of paper), but their descriptive power is poor when more abstract concepts are being represented, such as validating, linking and sorting. Finally, icons are useful as long as they are individual and unambiguous. When icons are used to represent several similar items, for instance entities in a database, the designer soon comes up against the limitation of the number of discriminable and meaningful pictures which can be created.

Advantages

Very easy to learn
 Easy to use
 Language independent
 Relatively easy to program

Disadvantages

Not economic on space
 Need some text backup
 Require graphics hardware
 Need icon builder software

Suitable for: novice users in system access and command interfaces. Computer-initiated dialogue.

(d) Form filling

Form filling is the most commonly used dialogue type for data entry but it also has uses in data retrieval and editing. The essence of the method is displaying a form on the VDU which is similar to the layout of a paper form with which the user is familiar (see figure 3.11). The display has a form title, prompts for the various fields, markers to show where the data should be entered and messaging areas to guide the user. The cursor is software controlled to move from one field to the next, either automatically or by using the Tab or Carriage return key. Data can be retrieved, displayed and edited after entry using the same display.

Forms have the advantage of a familiar layout, even if the form does not exactly model a previous paper document which users are familiar with. All the information is shown on the VDU and as long as the form is well designed, the sequence of operation should be self-explanatory. In data entry dialogues, form filling is accompanied by on-line validation and editing of the data. With a few minor design changes form filling is a suitable technique for data entry for both expert and novice users.

Advantages

Quick to use
 Easy to use
 Easy to learn

Disadvantages

Primarily suitable for data entry
 Unsophisticated

Suitable for: all user types, data entry, display and retrieval interfaces. Computer-initiated dialogue with some user control.

Ready	ORDER ENTRY		Date 12/12/87
Customer code	<----->		
Name	<----->		
Address	<----->		
	<----->		
Postcode	<----->		
Previous order	<_ _/ _ _/ _ _>		
	Catalog No	Quantity	Unit price Sub-total
Item 1	_____	_____	_____
Item 2	_____	_____	_____
Item 3	_____	_____	_____
Item 4	_____	_____	_____
Press TAB to move to next field			
ENTER to save			
E to exit			
C to change record			

Figure 3.11 Form-filling interface for a mail order system.

(e) *Command languages*

This is a large and varied category which covers single word command strings to complex command languages with a grammar. The common feature of command languages is that little or no supportive information is displayed for users who enter commands in locations indicated by prompts which are often a cryptic \$ or * symbol. The command then invokes a system operation which the user requires; when the operation is complete the command prompt returns. Because little information is displayed, command languages are very economical in use of screen space and, by the same token, in data transmission.

The great advantage of command languages is the sophistication and flexibility of the interface. If a system has a large number of functions which the user wants to access and, in addition, those functions may be required in different combinations, then a command language is the ideal interface. This is because various functions can be combined together in sentences using a grammar. Once a grammar is present, complex sentences can be constructed using the principle of nesting complex commands in phrases and substituting complex commands as a sub-routine identified by a simple name. In this manner the interface becomes a sophisticated and extensible method for controlling a system.

The penalty of command languages is that users have to learn a code and some form of grammar; this takes time and makes command languages difficult to use for beginners. The user also has to have some knowledge about what the system does because no information is displayed on the screen.

A further disadvantage is the development effort required for command language interfaces. Simple command interfaces can be implemented using keywords or a code set in which case only lexical checking is required to validate the command against a look-up table. As soon as the language has a syntax, a parser has to be built to check and interpret input. This becomes an increasingly demanding task which, taken to its logical extension, becomes compiler writing for programming languages. Command languages, however, come in various grades of complexity. Command such as the page address system in Prestel (that is, typing 134 accesses page 134 in the database) are simple. Single keyword command languages can also make a simple, easy-to-use interface. However, to realise the power of a command language a grammar is required, which makes this style more complicated for users.

Advantages

Quick to use
Sophisticated
Extensible

Disadvantages

Difficult to learn
Difficult to use for novices
Difficult to program

Suitable for: expert users with complicated command interfaces.
User-initiated and controlled dialogue.

(f) *Natural language*

Natural language should be the ideal human-computer interface because it is the user's natural method of communication. Unfortunately it has limitations from the user's viewpoint and poses considerable computational problems. Natural language may be input either directly as speech or

via a keyboard. Spoken input is quick and should be ideal, but the problems of deciphering speech are enormous and limit current speech recognition systems to simple phrases and single words. Typed input is verbose and time-consuming for the average user who is not an expert typist and prone to make typographical errors.

The major problems for natural language understanding is that meaning is generated at several different levels. First there is syntax which dictates how correct sentences should be formed; but to derive true meaning we need a framework of knowledge about words, their meanings and relationships. This information brings in the 'semantic' level of interpretation. Unfortunately this is often not enough because the meaning of a word can depend on the context in which it was uttered. To build a true natural language interface necessitates making computers mirror this process; this implies building a machine with artificial intelligence comparable to human intelligence with a vast database of word meanings.

Not surprisingly, natural language interfaces are currently practical only in limited domain problems. By limiting the domain, the quantity and complexity of knowledge required can be restricted to manageable proportions. Some limited natural language interfaces are practical for databases in which the interface has knowledge about the data items and a restricted set of linguistic terms which the user may employ to ask for the data.

Advantages

Natural communication
No learning required

Disadvantages

Difficult to program
Needs knowledge base
Verbose input
Can be ambiguous

Suitable for: novice and naive users in a restricted problem domain.
User-initiated dialogue.

3.7 Review of the Type of Interaction

The input for this step comes partly from interface analysis and partly from mainstream systems analysis as the functional specification of what data and input messages have to pass across the system interface. In most information and transaction processing systems this will be data and control messages specified as data flows (if structured analysis is being used) or data structures held in a data dictionary. Requirements analysis provides specification of the volume of data, the frequency of input/output, the timing (on demand or batch) and the validation constraints.

In some systems, requirements may have a more direct bearing on the dialogue type. For instance, a VLSI circuit design system may require an

interactive graphic interface with an icon library of circuit components. The functionality of a system can dictate the range of interface design types that it is possible to employ; consequently, it is necessary to classify tasks according to their type of interaction. Tasks can then be matched to interface design styles, bearing the user characteristics in mind. The task interaction type may or may not indicate a particular interface design style. A working classification of interaction types, which can be allocated to tasks, is as follows:

- Data entry
- Data display
- Data retrieval (search and display)
- Data editing
- Command (user access to system facilities)
- Conversation (series of questions and answers)

Although these categories are not mutually exclusive, they do give general guidance towards design of the interface. A command type of interaction may become a conversation if there is a long sequence of two-way dialogue between user and computer rather than short dialogues in which the user asks the computer to run a task. Menus or a command language are suitable for the latter while question and answer style is better for conversations. In a system with mixed types of interaction, a command language could be used with question and answer dialogues.

In some cases the influence of requirements will be strong, for instance a medical interrogation system which asks patients about their medical history will probably have to be a conversation; while a business graphics system will require a graphical display. However, it is important to remember that types of interaction do not pre-determine the dialogue style absolutely; a data entry function could be implemented by form filling, menus, touch panels or graphic-icon selection.

3.8 Selecting the Interface Design Style

This step aims to synthesise interface and systems analysis by taking the task descriptions, interaction types and the user profile, and arriving at a strategic decision about what style, or more likely styles, of interface are to be employed.

Interfaces should serve people; therefore, when selecting the type, human requirements come first, followed by system requirements. The decision steps are:

- From the user profile, decide on the level of support and sophistication which the dialogue should provide

- Select one or more interface styles which are appropriate for the support and sophistication required
- Match the interface tasks against the system requirements and categorise the type of interaction
- Select appropriate interface design styles. If there is a clash between user and system requirements, then trade-off decisions will have to be taken

The user profile specifies the level of support needed and degree of sophistication. These measures can be derived from a simple table in which the familiarity, frequency of usage, general ability and user's knowledge of computer operation are scored on a simple scale of +++ = good/frequent to --- = poor/infrequent, and matched against the sophistication and support which are desirable:

<i>User measures</i>		<i>Interface characteristics</i>	
Familiarity	++	Sophistication	++
Frequency	++	Support	-
Ability	++	(expert interface)	
Knowledge	++		
Familiarity	++	Sophistication	++
Frequency	++	Support	+
Ability	++	(skilled user interface)	
Knowledge	-		
Familiarity	-	Sophistication	+
Frequency	-	Support	++
Ability	++	(skilled novice)	
Knowledge	-		
Familiarity	+	Sophistication	-
Frequency	++	Support	++
Ability	-	(dedicated unskilled)	
Knowledge	-		
Familiarity	-	Sophistication	--
Frequency	-	Support	+++
Ability	-	(naive user interface)	
Knowledge	-		

Highly skilled expert users who use a system frequently will require a sophisticated interface to fulfil the complicated functions which they wish

to undertake and to give them the ability to extend the system's properties to suit their own needs. Users who are skilled and use the system frequently but lack background knowledge about the system structure are less likely to need the flexibility to extend the system's powers, even though they will still require a quick-to-use and sophisticated interface. High-frequency users who lack knowledge about the system structure and have low to moderate ability are unlikely to be able to deal with sophistication and need a supportive but quick-to-use interface. The inverse user profile (high ability, low frequency) is suitable for a sophisticated interface but will require a high level of support because the low frequency of use will lead to the interface characteristics being forgotten.

An approximate guide to link user abilities to interface type is:

<i>Type</i>	<i>User abilities</i>
Expert user Interface	Programming language Extensible command language
Skilled user Interface	Command language Code-keyword interfaces
Skilled novice	Code-keyword interfaces Menus
Unskilled dedicated user	Menus Page address
Naive user	Question and answer Simple menus

Inevitably, users in a population will rarely fit neatly into one category so the eventual choice depends on trade-off decisions which try to satisfy as many different types of user in the overall population as possible.

The level of sophistication can point strongly to a dialogue type; at this stage the system requirements are introduced to home-in on a small range of types. Support has a less direct bearing on the dialogue type. Obviously a requirement for a very supportive dialogue should not be implemented with command languages; however, a sophisticated dialogue can be successful with fairly inexperienced users if good support is provided in terms of help screens, tutorial guides, training and documentation.

Interface styles frequently have to be chosen within the limitations of the available hardware which may constrain some more innovative solutions. The usual hardware encountered is the ubiquitous VDU; in this case the major constraint on interface design is provision of high-resolution screens; although with the growth in high-resolution raster graphics, this constraint should become less important in the future.

Finally, consideration of the user's views may have implications for the choice if there is a strong structural metaphor in the view. If the user views the system in terms of a map or a spatial linked collection of objects, then use of graphics and icons would be advisable.

Once all these factors have been considered and interface design styles have been chosen, the way is open to the next step of interface design which is carried out using the properties, and within the limitations, of the chosen interface style. This step takes the task design, now tagged with design styles, and adds the dialogue to support the human use of the computerised tasks.

Synthesis of the products of analysis into design decisions is in the end dependent on experience. The above guidelines are intended to be a framework within which to work, with the aim of ensuring that at least all the factors pertaining to the decision have been examined.

3.9 Summary

Task analysis is similar to functional analysis as practised in systems analysis and design. Top-down functional decomposition is used to break tasks down into smaller components which can then be specified in detail. The techniques of Structured analysis, Data-flow diagrams and Structured English may be used for this purpose.

Besides task analysis, analysis of user characteristics is important. Qualities of frequency of use, general ability and computer experience contribute towards measures of user sophistication and support. These measures can then be used to plan the type of interfaces suitable for a user population. Users can be approximately categorised as Naive, Novice, Skilled and Expert, depending on their previous experience.

User models have several different objectives. User characteristics, user task models and user views are the more important models for interface design. User task models attempt to capture the user's knowledge about how a system is expected to operate while user views capture the user's perception of system structures. The closer an interface conforms to these pre-conceptions, the easier it should be to use.

General interface design starts with task design. Allocation of actions and tasks to either human or computer, or both, is the first step. Allocation is best carried out at the detailed level of actions. Joint human-computer tasks and actions may need further analysis. Task networks are drawn up for the human and computer system.

Task design then re-organises the human task network to create designs which allow for human limitations. The network is restructured to include closure events. Tasks are combined into work modules and jobs. Task combination aims to produce jobs which have the correct degree of

stimulation while not overloading the operator. Care must be taken to avoid task overload, especially with processing transactions peaks and error cycles.

System support design involves documentation and training manuals. Structuring the information and simple clear layouts are vital. Manuals have to support trouble shooting as well as education, and good access paths should be provided to information.

Interface design has a basic series of styles which consists of Question and answer, Menus, Form filling, Command languages and Natural language. System requirements constrain the choice of dialogue style to an extent and have to be analysed by matching requirements against suitable interface styles. User characteristics are the more important determinant of which style is eventually selected. The interface design style is matched against the projected user sophistication. The user support requirement measure determines the level of environmental support which will be necessary.

Further Reading

Further details of task analysis and job design can be found in Bailey (1982) and Damodaran *et al.* (1980).

4 *Theoretical Approaches*

So far, a general method of interface design has been presented which is based on the concepts of structured systems analysis and design. Other methods of interface analysis and specification devised by workers in human-computer interaction are now presented for a comparison. Two main groups of methods have evolved within human-computer interaction: grammatic and diagrammatic techniques. Of the grammatic school, the best developed and well known method is the Command Language Grammar (CLG) of Moran (1981).

4.1 **Command Language Grammar**

Like structured analysis, CLG employs top-down functional decomposition to analyse systems; however, CLG has several levels which start from an analysis perspective and progress towards a physical design. This approach puts analysis in a framework of abstraction from the goals of what the interface has to achieve through to the detail of how the interface will operate. CLG uses a semi-formal language of reserved words for structure and sequencing, indentation to show the scope of control, verbs for actions and nouns for data. The reserved word set is not completely specified, making it open to extension by users. The CLG specification levels are:

- *Task level*: this analyses user needs and how those needs should be achieved in terms of goals and sub-goals. User tasks are described using English narrative in terms of objectives and goals
- *Semantic level*: this level elaborates the system as a set of objects and operations carried out on those objects. Tasks are formulated in terms of conceptual entities, operations and methods which specify how operations are organised
- *Syntactic level*: operations of the semantic level are refined into a language composed of commands, user operations, contexts and state variables. It describes how the user and system components interact according to the grammar
- *Interaction level*: this level defines user operations and system commands in terms of physical operations such as keystrokes, device manipulation and displays

As CLG is a grammar, it is composed of expressions which obey certain rules. The general format is:

Symbol (arbitrary name) = expression list

Three expression types are defined. First is a hierarchic expression used to describe objects (for example, message = a memo); this expresses the classification 'a memo', being an instance of the class message. Second is a set or sequence membership expression (SET = x,y,z). The third is ordinary English narrative. Expressions describe concepts and can be represented by an identifier; in this way sub-expressions and hence concepts can be embedded within other expressions in a hierarchical manner.

At the task level CLG describes concepts in English. Tasks may be divided into a hierarchy with sub-tasks and so on (see figure 4.1). Each task and the entities upon which they act are described in a structured format with notes on organisation and any constraints. The example is an E-Mail system, described in full in Moran (1981):

SEND-MESSAGE = (AN ENTITY NAME = "Send Message")
 (* this is a message sent by the SEND system
 A SEND message has a header and a body
 The header contains . . .)

NEW-MAIL = (A TASK (* Check for new SEND MESSAGES and if any read them. This is the most frequent task))
 DO (SEQ: (CHECK-FOR-NEW-MAIL)
 (READ-NEW-MAIL))

READ-NEW-MAIL = (A TASK (* Read all new SEND MESSAGES, deleting all those that are of no further interest))

The major difference with Structured English is the declaration of data objects, called *entities*. These are data aggregates upon which actions happen. The data is related to a common something for which no definition is given; however, in practice entities can be thought of as objects of interest in the system. Top-down decomposition continues until detailed assumptions have to be made about the structure of entities and tasks; this point is not clear, but it is closer to the action than the function concept in structured analysis.

At the semantic level CLG defines conceptual entities and operations which will carry out the tasks. Some English narrative is still used but more detail is included for entities and tasks. In the messaging system the semantic statements may be:

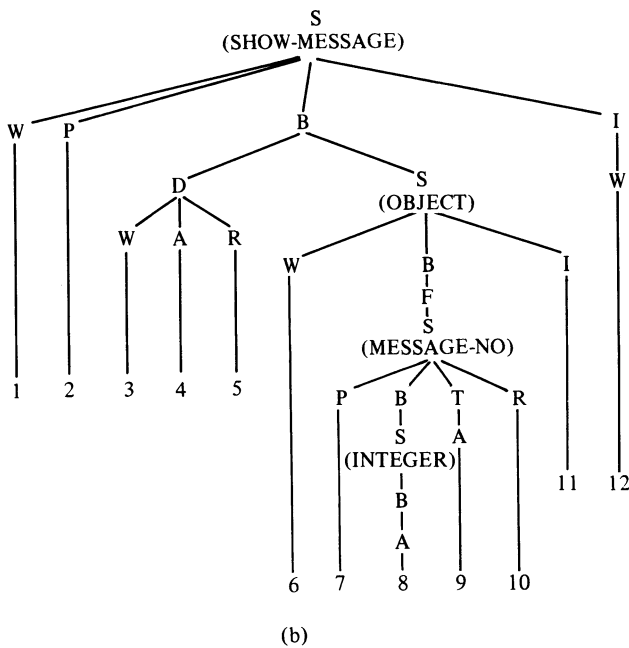
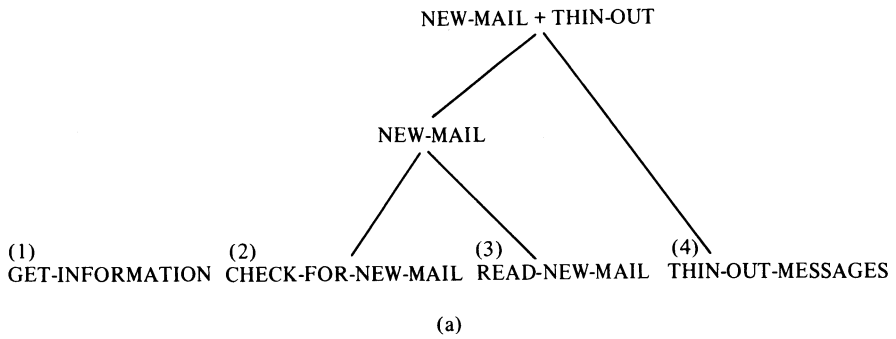


Figure 4.1 The Command Language Grammar: (a) at the task level showing a task hierarchy for an electronic mail system; (b) at the interaction level showing the tree of possible interactions for one command, Show message.

MAIL-SYSTEM = (A SYSTEM NAME = "Email"
 ENTITIES = (SET: MESSAGE SUMMARY MAILBOX
 SCREEN DIRECTORY)
 OPERATIONS = (SET: SHOW DELETE))

MESSAGE = (AN ENTITY REPRESENTS (A SEND MESSAGE)
 NAME = Message
 (* narrative description . . . message has a header and a
 body etc. *))

MAILBOX = (A LIST
 REPRESENTS (A MESSAGE FILE)
 OWNER = (A USER)
 MEMBER = (A MESSAGE)
 NAME = "Mailbox"
 (* this contains all messages . . .))

Conceptual entities may be pure concepts, such as a Mailbox, or more tangible objects such as a Message. Hierarchical structuring can be shown in LIST Entities which group other entities; the system itself is composed of a SET of entities. Entities are described in this manner with various properties such as owners, membership of sets or name identifiers. Operations are linked to objects and values which are necessary to satisfy their function. Each operation may be described further in narrative or may not require elaboration. The difference between operations and tasks is not made explicit in the method. Operations, conceptually, happen to objects within the system, while tasks are a more goal-oriented statement of the user's intentions. In practice the two become merged as analysis progresses. Operations are classified as User or System operations.

SHOW = (A SYSTEM OPERATION
 OBJECT = (A PARAMETER VALUE = (AN ENTITY))
 IN (A PARAMETER VALUE = A PLACE ON THE
 SCREEN)
 (* narrative description . . . object may be a MESSAGE,
 SUMMARY OR DIRECTORY *))

READ = (A USER OPERATION
 OBJECT = (A PARAMETER
 VALUE = (AN ENTITY))
 IN (A PARAMETER
 VALUE = (A PLACE ON SCREEN)
 DEFAULT-VALUE = (UNKNOWN)
 (* The User reads an Object, which is in some place on the
 Screen. The Object may be a MESSAGE, SUMMARY or
 DIRECTORY)))

The last component of the semantic level is a method or procedure for achieving a task. Methods add control constructs to operations and entities of the form 'Do while' and 'If then else'. If the reader perceives some similarity with the GOMS model described earlier, that association is correct; CLG owes much of its heritage to cognitive problem-solving models. An example of a method is:

```
SEM-M1 - (A SEMANTIC METHOD FOR CHECK FOR NEW MAIL)
      DO (SEQ: (START SYSTEM)
          (SHOW DIRECTORY)
          (LOOK AT DIRECTORY FOR
            (A MESSAGE = NEW)))

SEM-M2 = (A SEMANTIC METHOD FOR READ-NEW-MAIL)
      DO (REPEAT
          BINDING M TO (EACH MESSAGE
            AGE = NEW)
          DOING (SEQ: (SHOW m)
                  (READ m)
                  (OPT DELETE m)))
```

The binding operation instantiates a variable in an entity. The method then shows the procedural sequence of operation in terms of the three basic control structures SEQuence, REPEAT and OPTional selections. At this stage the rules governing selection and repetition are not detailed.

At the syntactic level, operations and methods are defined as commands which are issued by the user to the system. Commands are created for each semantic operation. Syntactic commands refer to entities with arguments and have contexts, that is, situations in which they may and may not be used. One or more contexts describe the whole system in terms of displays, commands and state variables which hold values within a context and are used for control (such as Message numbers). The system Entities are refined at the syntactic level into more physical objects which will correspond to screen displays. This gives a more detailed description of their properties, for example, message layout, screen display areas, directory structure, etc. Also added at this level are the identifiers to be used by the user to find conceptual entities, called Descriptors. In the Mail system these are simple message numbers.

```
MAIL-CONTEXT = (A COMMAND-CONTEXT)
      STATE VARIABLES = (SET: CURRENT MESSAGES)
      DESCRIPTORS = (SET: MESSAGE NO)
      DISPLAY AREAS = (SET: DIRECTORY AREA,
                      MESSAGE AREA, COMMAND AREA)
      COMMANDS = (SET: SHOW-MESSAGE SHOW-NEXT,
                  DELETE-MESSAGE EXIT)
```

```

MESSAGE NO = (A DESCRIPTOR
              NAME = "Message number"
              FORM = AN INTEGER
              VALUE = (A MESSAGE)
              DEFAULT VALUE = (THE-CURRENT-MESSAGE))

```

```

MESSAGE AREAS = (A-DISPLAY-AREA)
                 NAME = "Message Window"

```

Descriptive narrative may also be included in the syntactic level to clarify the links between display areas and the objects to be displayed. At the syntactic level, conditions are added to repetitions and options in Methods. The show command becomes:

```

SHOW MESSAGE = (A MAIL-COMMAND)
               NAME = "message"
               OBJECT = (AN ARGUMENT = (MESSAGE NO))
               DOES (SET: (SHOW (SUMMARY OF (THE OBJECT)))
                    IN DIRECTORY AREA
                    (SHOW (MESSAGE NO OF (THE
                    OBJECT))) IN DIRECTORY AREA
                    (SHOW (THE OBJECT)
                    IN THE MESSAGE AREA))

```

```

DELETE-MESSAGE = (A MAIL-COMMAND)
                 NAME = "delete"
                 DOES (SEQ: (DELETE (THE CURRENT MESSAGE))
                      (IF (THERE-IS (A MESSAGE) IN
                      MAILBOX)
                      THEN (SHOW-NEXT-MESSAGE)
                      ELSE (SEQ: (DISPLAY (* No more
                      messages)) IN COMMAND AREA)))

```

The syntactic level brings the specification down to the detail of what commands can be used in the system, the effects the commands have on the objects in the system, the messages displayed and the screen layout of displays for messages and objects that the user needs to see. Syntactic methods are similar to the semantic level although the specification is refined to include more detail of system operations, for example:

```

SYN-M2 = (A SYNTACTIC METHOD)
         FOR READ-NEW-MAIL
         DO (REPEAT UNTIL (* End of mailbox)
         DOING (SEQ: (READ (THE-CURRENT-MESSAGE)
         IN MESSAGE-AREA)
         CHOICE: (SHOW-NEXT-MESSAGE)
         (DELETE-CURRENT MESSAGE)))

```

The show command in the user's task can now be achieved by the system commands which have been designed for it.

More detail is added at the interaction level which describes the dialogue and presentation design. In this step CLG becomes a true grammar composed of terminal symbols (which are self-defining and cannot be subdivided) and non-terminal symbols which are composed of terminal symbols. The terminal symbols are:

W	When is	(temporal specification primitives—before, after)
P	Prompt	} primitive system action—display (x)
R	Response	
A	Action	(primitive user action—a keystroke)

These can be combined into non-terminal structures. Interaction is described as a tree for each command operation, the tree defining the permissible sequences of prompts, responses and states. Inspect figure 4.1b and cross-refer to the text below:

2 (P.S	OF SHOW-MESSAGE—(DISPLAY “Command”) {where S = the specification})
4 (A.D.B.S	OF SHOW-MESSAGE—(KEY: “M”) {where D = “the designation of” B = “the body of” S = “the specification of”})

The interaction level also elaborates methods by adding validation of interactive commands which may be used, actions to be taken in response to commands, and specifying messages, for example:

```

MX = (AN INTERACTION METHOD FOR READ NEW MAIL)
      DO (REPEAT UNTIL (* End of mail))
      DOING (SEQ: (READ (THE CURRENT MESSAGE) IN
                    MESSAGE AREA))
              (CHOICE: (KEY: “N”)
                     (KEY: “D”))

```

Rules are added to link commands with states (When), prompts and responses. The rules are fairly simple for single commands but if commands have arguments then further rules are required and complexity increases.

The above description is just an overview of CLG. For more detail the reader is referred to Moran (1981), although the present description should be sufficient to appreciate the essentials of CLG. First it is a hierarchical analysis and specification from a conceptual dimension to the detailed level of physical interaction. There are informal mapping rules between the levels, for instance:

Entities:	Task conceptual entities—semantic system entities—syntactic descriptors
Tasks:	Tasks—semantic procedures and methods—syntactic procedures and methods
Operations:	Semantic operations—syntactic commands

Whether mapping is 1:1 or not is open to the analyst's discretion.

Task design occurs primarily at the lower levels as the detailed specification is developed. The task level describes the user's requirements as a set of goals and informal task descriptions; the semantic level follows this by describing the functionality of the system; and the syntactic level adds design detail of how the functions are evoked. Finally, the interaction level specifies the physical form of the command language and dialogue to support the tasks. Mapping between the levels is not always explicit and considerable judgement has to be exercised by the analyst. Also, CLG's critics may regard it as cumbersome and over-detailed especially at the interaction level. Nevertheless, CLG does form a powerful specification and design method.

4.2 Other Grammatical Specifications

One approach using Backus Naur Form (BNF) to notate a task-action grammar has been pioneered by Reisner (1984). Her objective was to create a predictive analysis of command languages by comparing the complexity of languages in terms of a metric derived from the grammatical specification. The metric is based on the number of command words and grammatical rules as expressed in Backus Naur Form. As a result the scope of Reisner's BNF specifications is not as comprehensive as CLG and expert user knowledge of the command language syntax is assumed when assessing the language. The basic method is to describe all valid commands in grammatical terms composed of terminal and non-terminal symbols.

Terminal symbols are the basic words of command language; these are combined into non-terminal symbols, alias phrases, clauses and sentences. In addition there are a few special symbols for notational purposes such as ::= 'is composed of', + 'and', | 'or'. Complexity is built up by nesting smaller components within larger ones. In natural language this can be seen in paragraphs which are composed of sentences which in turn are composed of phrases, etc.

Grammars of this sort can be used to specify interfaces at different levels. To compare interface designs the word set is composed of verbs for physical actions that the user can perform, such as point, enter keystroke, position; and nouns describing the interface objects such as the cursor, display, key. By comparing the complexity of the grammatical strings and

the number of terminal symbols, a judgement could be made about the complexity of the command language. Generally the less complex a command language the better.

Another extension of grammatic techniques is to tag the phrases as being (1981) which uses GOMS as its basis and then adds set times for various cognitive operations into which tasks are decomposed. Operations are classified according to components of the information-processing model, for example reads and writes to short-term memory, perceptual actions and mental action cycles of the cognitive processor. The method works by describing the task as sequences of these primitive cognitive actions, assigning times to the actions and thereby deriving an estimated task completion time. Unfortunately the method assumes error-free performance which makes its use questionable.

Another extension of grammatic techniques is to tag the phrases as being performed either by humans or computers (see Shneiderman, 1981). This enables sequences of interaction between human and computer to be described at the task level. An alternative approach has been to use a generalised task-action grammar to describe conceptual objects and actions thereon in specific task domains. This method, TAKD (Johnson, 1985), creates a generalised task model by abstracting from specific objects in a domain (a letter in a word processor) to conceptual objects in the system (documents). The generalised task model is then mapped on to a generalised system model which specifies the system objects (which become data structures) and actions (which become dialogue).

In conclusion, grammars have been used at different stages in interface development with the motivation of either early evaluation by predicting qualities of design or for describing and analysing interaction.

4.3 Diagrammatic Specifications

Diagram-based interface specifications have employed state event transition diagrams and occasionally data-flow diagrams for task and dialogue description. The latter have already been described, so attention will be focused on the use of state event transition diagrams.

State event transition diagrams, also known as *finite state machine diagrams*, are familiar throughout computer science as a method of describing sequences of events within a system. The components are a *state*, represented as a circle, and a *change of state*, otherwise called an *event*, shown as a connecting arc. The diagrams may show branching to account for divergence in a sequence, and repetition of a state event cycle. However, state transition diagrams are not suitable for showing concurrency and hierarchical structures. These facets are necessary to specify operations in concurrent windows within an interface and to control

complexity in specifications. Most authors have added extra features to deal with the main defects. These diagrams are more useful for detailed design of dialogues and are described further in chapter 5. The most influential method in the diagrammatic camp is Cognitive Complexity Theory of Kieras and Polson (1985). As its name implies, this theory specifies interface complexity, besides creating a diagrammatic specification of dialogues.

4.4 Cognitive Complexity Theory

Classifying CCT as a diagrammatic method is not truly accurate. The method uses two formalisms, production systems and generalised transition diagrams, to specify in turn the user's knowledge of the task and then the user-system dialogue. The main focus of this method is analysis of task complexity, so tasks and hence interfaces can be designed which do not overload the users' capabilities. The task is analysed using production systems to describe the user's model of the task in terms of what is known about how to do the task. Production systems are rules in the form 'IF condition THEN do action' with associated working memory holding facts to be evaluated in the condition.

The method aims to analyse the task fit between the user's concept of the task and how the task model is formulated in the system. CCT distinguishes between device-dependent and device-independent (that is, pure task) knowledge. Complexity is considered to be caused by:

- Complexity of the user task in terms of learning and memory load
- The number of device-dependent functions which have to be learned
- The ease with which 'how to work it' knowledge can be acquired

The better the task-system fit the smaller the number of device-dependent functions should be, and the more natural a design will be to use.

The production systems are arranged in procedural sequences of task actions and the human reasoning behind the task, which is expressed in a GOMS-like goals network. Goals can be added, changed or deleted by the productions and this allows a description of the traversal of a problem-solving network to be made. Actions allow goals to be inserted and deleted from working memory as well as performing manipulations on the environment. The production systems proceed by alternate recognise (test condition) and act modes.

Production system sequences form methods for achieving goals and selection rules test facts in working memory and control the execution of methods and actions. Special conditions are added to evaluate the presence and status of goals in working memory as TEST GOAL and ADD NOTE (status variable). Finally there are variables notated with a % which can

become instantiated with values. The whole system has a hierarchical organisation. An illustration of a production system for an editing task is:

```
(task edit article
  IF(and(TEST-MSS manuscript is new article)
    (TEST-GOAL type manuscript)
    (TEST-GOAL select equipment)
  THEN ((ADD-NOTE many revisions will be done)))
```

This is a top-level goal establishing the nature of the task. Lower-level goals and the associated methods are:

```
(SET UP-UNIT-TASK
  IF(AND(TEST-GOAL edit manuscript)
    (NOT (TEST-GOAL perform unit task)))
  THEN ((GET-NEXT-UNIT-TASK)
    (ADD-GOAL perform unit task)))
```

Unit tasks are selected in sequence to effect completion of sub-goals. In the word-processor example, goals are broken down into editing operations of decreasing complexity and finally into unitary simple operations such as deletion, insertion, replace, etc.

Some selection and control rules for deletion:

```
(SELECT-CHARACTER-DELETION
  IF(AND(TEST-GOAL perform unit task)
    (TEST-MSS function is delete)
    (TEST-MSS entity is character)
    (NOT (TEST-GOAL delete character)
    (NOT (TEST-NOTE executing character deletion))))
  THEN ((ADD-GOAL delete character)
    (ADD-NOTE executing character deletion)
    (LOOK-MSS task is at %UT-HP %UT-VP))))

(CHARACTER-DELETION-DONE
  IF(AND(TEST-GOAL perform unit task)
    (TEST NOTE executing character deletion)
    (NOT (TEST-GOAL deleting character)))
  THEN ((DELETE-NOTE executing character deletion)
    (DELETE-GOAL perform unit task)))
```

The method for deleting a single word, PDELWD1, is illustrated in the following text. First the cursor is positioned, then the word is deleted and finally the goal is removed from memory.

```

PDELWD1
IF(AND(TEST-GOAL delete word)
      (NOT(TEST-GOAL move cursor to %UT-HP %UT-VP))
      (NOT(TEST-CURSOR %UT-HP %UT-VP)))
THEN (ADD-GOAL move cursor to %UT-HP %UT-HP)

PDELWD2
IF(AND(TEST-GOAL delete word)
      (TEST-CURSOR %UT-HP UT%-VP))
THEN ((DO-KEYSTROKE DEL)
      (DO-KEYSTROKE SPACE)
      (DO-KEYSTROKE ENTER)
      (WAIT)
      (DELETE-GOAL delete word)
      (UNBIND %UT-HP %UT-VP))

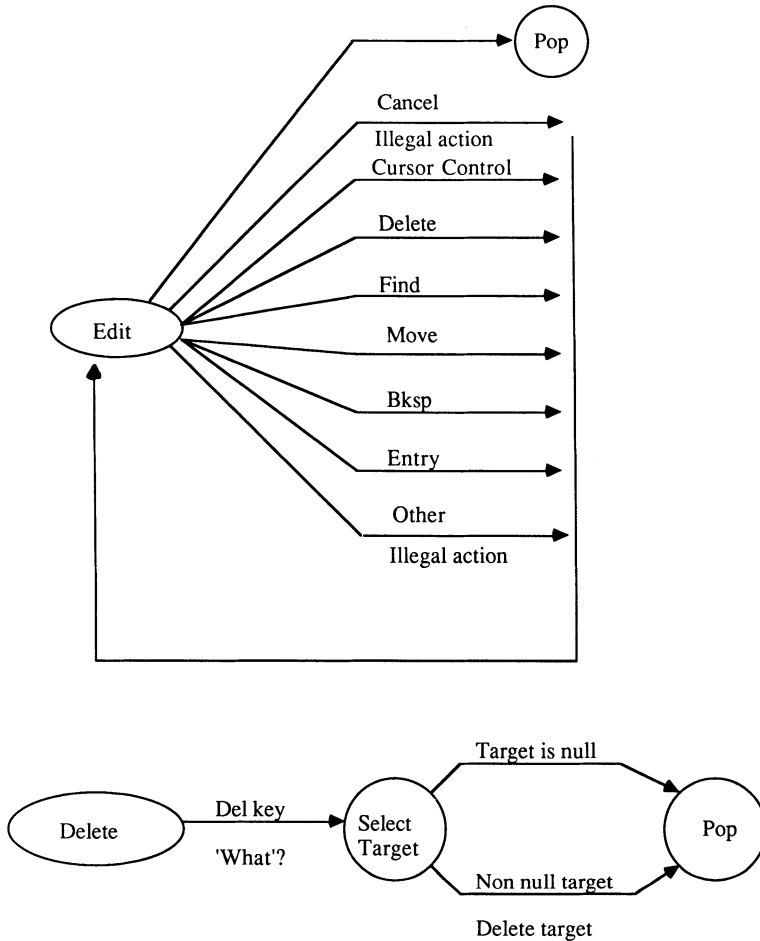
```

The production systems describe the goal manipulations necessary to control operation of the task; the testing of external variables and the update of status variables are operations progress. All these impose a load on working memory which can be quantified. The number of production system rules can be counted; more rules make a system more difficult to learn.

Another analysis is to examine the goal structure in the system. There should be only one goal structure per task, any more will confuse the user by presenting unnecessary complexity with many ways of doing one task. Also the system goal structure can be compared with the user's model of the task goal structure to test for goodness of fit.

The second part of Kieras and Polson's method uses generalised transition network (GTN) diagrams to model the device dialogue. GTNs are derived from state transition diagrams augmented with a hierarchical nesting feature to deal with complexity. Hence top-level diagrams call sub-diagrams. The components of GTNs, illustrated in figure 4.2, are states (prompts and computer displays) shown as circles, transitions between states caused by human actions and replies, illustrated as arcs, and conditions which control the transitions. Nesting can occur in conditions, states or actions. Diagrams read from left to right, and conditions/action arcs are positioned clockwise around a circle state in the order in which the conditions are tested. Nesting is shown by sub-network calls and POP for the return-exit point in the sub-network.

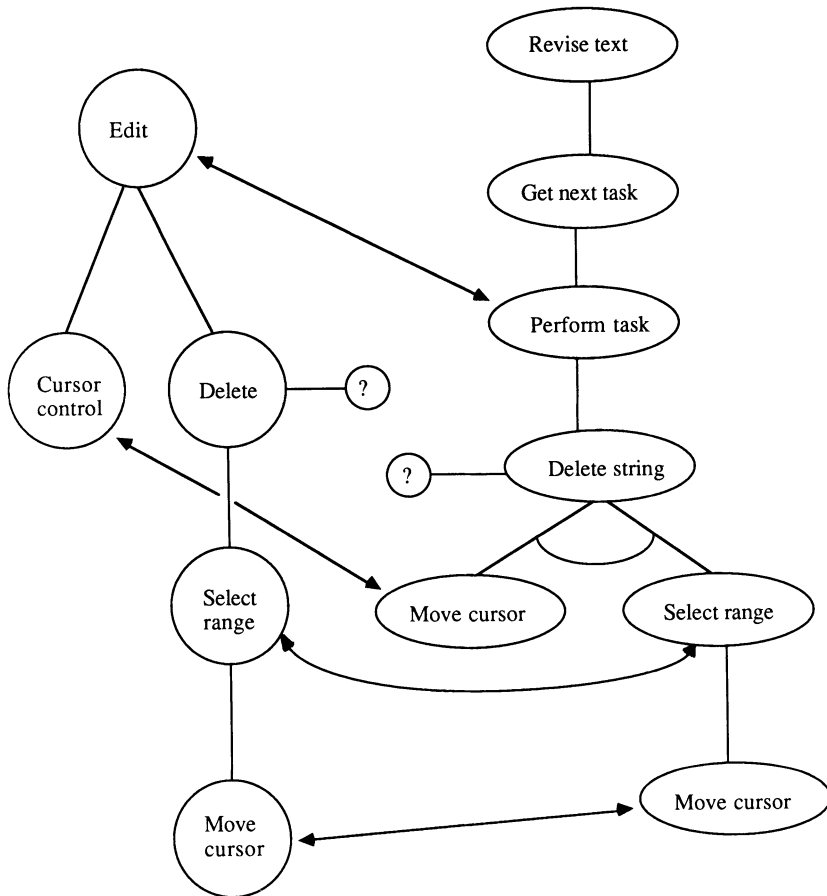
The nesting hierarchy of GTNs represents the system goal hierarchy. Hence the user and system goals hierarchies can be compared to discover the extent of task-tool fit. Drastic divergence in the two hierarchies indicates that users will have problems using the device because of a poor match with their conceptual model. An example of poor user goal-device hierarchy matching is shown in figure 4.3.



(after Kieras and Polson 1985)

Figure 4.2 Generalised transition network diagrams used for dialogue design in Cognitive Complexity Theory. The system illustrated is a word processor.

As well as providing a method for examining complexity in tasks and the mapping between user and system task models, the GTNs in CCT are a useful method of dialogue design which can be used to plan and verify good practices of dialogue design. The method is used for this purpose in chapter 5.



(after Kieras and Polson 1985)

Figure 4.3 Mapping between the user conceptual model and the system model. Some user goals have no corresponding match in the device model, leading to poor task-tool mapping.

4.5 Comparison of Specification Methods

Ideally, human-computer interface specification and design methods should cover all aspects of the interface development life cycle, be easy to use and learn, have predictive qualities for evaluating designs, and cover the diverse aspects of specification (cognitive load, task model, system

model, dialogue and presentation design). This is a tall order which no method has so far addressed.

Some commonalities in approach, however, can be discerned. Many methods make reference to levels of abstraction. Commonly used terms are the Semantic (what to do), Syntactic (how to do it) and Lexical (physical details) levels, for instance, see Foley and van Dam (1982) and Shneiderman (1987). There is agreement that task-semantic specification occurs first and is then mapped on to features in a design. Another theme is conceptual modelling, in particular discovering the user's conceptual model and then basing a design on it. Cognitive complexity theory and Command language grammar provide a mechanism for this.

Both CLG and CCT have been criticised as being too complex for practical use, although they have been used with a pragmatic reduction in complexity of notation. CLG also presents problems in the mapping rules between levels, which even though they are stated in Moran's paper are not sufficiently complete to guide the novice practitioner. Nearly all methods omit coverage of the early stages of task analysis and assume that the analyst has a clear picture of what the user wishes to do. As such, they are methods for describing the system and then designing the interface, and not complete analysis and design methods.

Another point of comparison is in the merits of notation; do grammars or diagrams make a clearer specification? Unfortunately there is no complete standard within either approach. Grammatical methods have the advantage of making the structure of a dialogue clear and specify permissible computer and human actions concisely. Grammars can be used in different levels of specification from the conceptual to physical domains; and they are flexible because word sets and grammatical rules can be adapted to suit the application and needs of the investigator. But that flexibility limits the utility of grammars as a general specification method. Each institution has to create its own set of words and grammatical rules; also, the mapping between grammar-based specifications at different levels is difficult to specify in a formal manner.

A disadvantage of grammars is the poor illustration of sequencing. Although sequences may be specified by arranging phrases in approximate order of occurrence or by tagging them in an interactive series, text-based methods obscure any complex sequencing. As most dialogues are networks, sequence information may be important when planning human-computer interaction.

Diagrams can be used in interface specifications as a recording medium for time-ordered tasks and dialogue designs. Whereas grammars are good for structure, diagrams are good for sequences and network specifications. However, when dialogue networks become complex with high connectivity, diagrams become more difficult to understand, and maintaining diagrams without automated support can be arduous.

4.6 Summary

Many methods of interface specification have been proposed by researchers in human–computer interaction. Two main groups are discernible, diagrammatic and grammatic approaches, of which Cognitive Complexity Theory and Command Language Grammar are the most influential examples in each category.

HCI specification methods use concepts of levels of abstraction to describe the user’s conceptual model at the semantic level, and then refine it into design of interaction at the syntactic and lexical levels. CLG is probably the most complete method although it suffers from over-complexity in notation. Diagrammatic methods can give clearer representations of task and dialogue as long as the connectivity is not too complex. CCT uses GTN diagrams to illustrate the user’s task and the user–system dialogue. In addition it employs production systems to analyse complexity.

Both diagrammatic and grammatic techniques have their merits; however, current HCI specification methods tend to be focused on a particular concern rather than covering all the issues in the lifecycle of development.

Further Reading

In addition to the references cited in the text, the INTERACT and CHI conference proceedings are a good source of HCI specification and design methods.

5 Dialogue Design

This chapter takes interface design from the strategic to the tactical level. The interface is designed first as a set of logical modules using input from task design, and then the modules are organised into an interface structure by addition of an access mechanism. Access mechanisms are the way in which users address data or functions provided by the system, and can be hierarchic, network or direct. The type of mechanism will be dictated mainly by the task structure and to an extent by the interface design style. For structured task systems, menus present a hierarchical organisation, while command languages provide for network and direct access, and icons are a direct access mechanism which may also have a hierarchical structure. The logical modules are then mapped on to physical screens, windows and overlays, depending on the target hardware and software environment.

Each module is decomposed into discrete steps, each step being a single question and answer between man and machine. The steps are then re-assembled into a detailed dialogue design which describes how the interface communicates with the user. The detailed design aims to incorporate good practices of dialogue design and provides some means of verifying that the design adheres to those practices. The steps involved are first designing the structure, then the access paths, followed by either prototyping the design with interface design tools (4th generation languages, screen generators) or detailed design of each step before implementation. The steps are summarised in figure 5.1 Which route is followed will depend on the complexity of the interface. More complex interaction in which the dialogue is critical should be designed in detail.

5.1 Designing the Interface Structure

So far the interface specification is composed of a set of task descriptions, system requirements and a strategic choice about the style of dialogue design which is going to be used. The next task is to add the access mechanism and then subdivide the whole interface into modules which can be mapped on to physical structures and then programmed as parts of the system interface, such as data entry screens, help and error overlays, reports, menus and command lines.

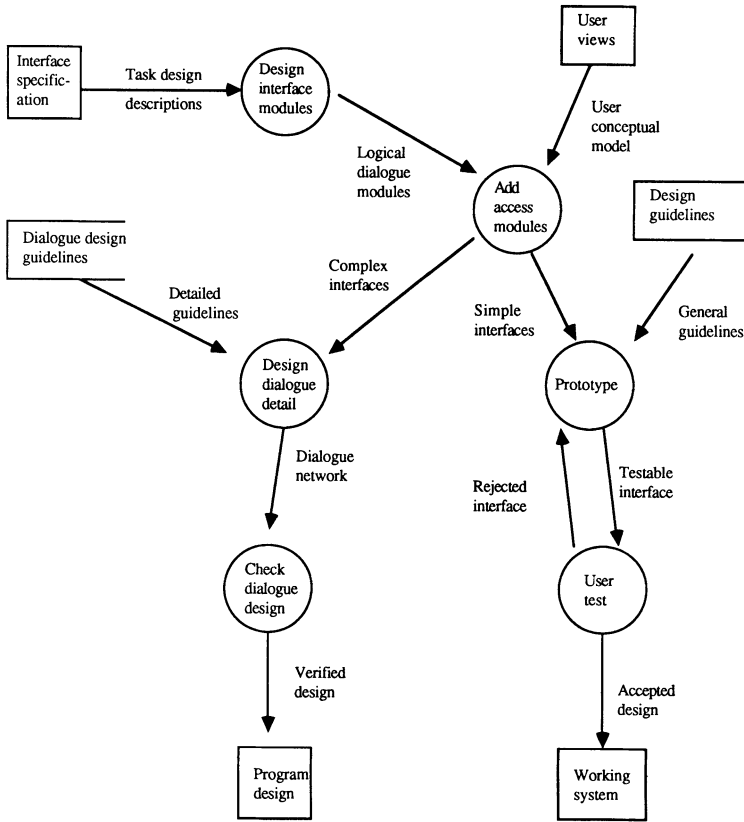


Figure 5.1 Flow diagram showing the steps of detailed interface design.

5.1.1 Adding User Access and Control

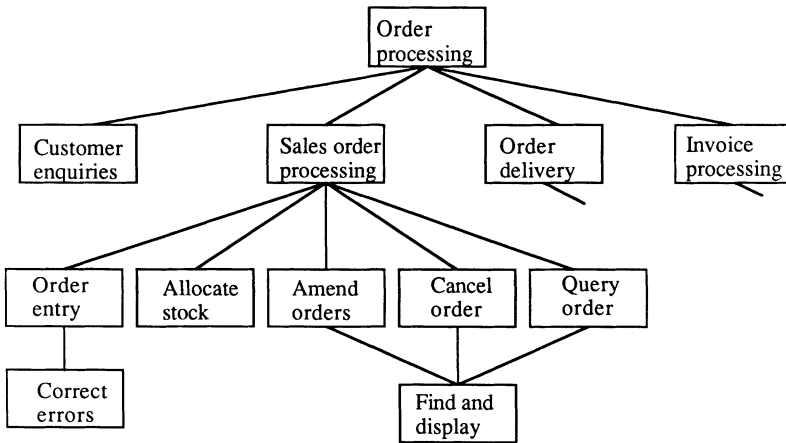
As user access is not part of the task description, it has to be added as part of the new system. The aim is to synthesise the user's view and the task structure. With luck these will agree, but sometimes the analyst may perceive a network of linked tasks which the user sees as a set of independent tasks. When in doubt the user usually knows best.

Interface modules will rarely be ordered in a simple sequence. The user's view or the user requirements for system operation may state that certain modules must be available on demand while others should be organised in sequences to achieve a particular task. Organising interface modules into a system depends on user characteristics, which influences the choice of interface style, the system-task structure, and the user view of the interface. Access paths may be hierarchic, network or direct. A menu style

will tend to create a hierarchic organisation for the interface, whereas command languages can provide either direct access or network association between interface modules. Depending on the type of interface chosen, the modules will be linked together either as a hierarchy or as a network, which can be shown diagrammatically, as in figure 5.2.

If hierarchical access is being used, how the modules are grouped together will be influenced by the user's view of the system as well as by the

1. Order processing, showing a hierarchical structure which may be implemented using menus.



2. Library system. This system has a network task structure

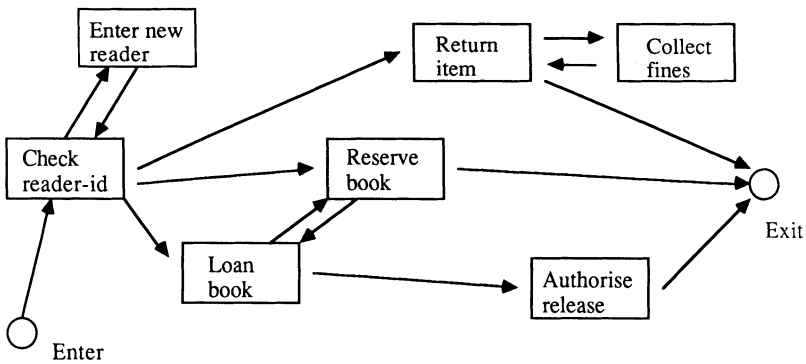


Figure 5.2 Interface structure diagram, showing modules and access paths for an order processing and library system.

functions which the system provides. Users may have several views which relate to different parts of the system. There may be a physical layout view and a logical classification view, the former expressed as a type of map and the latter as an abstract hierarchy. Views of a classification of objects are often expressed in terms of hierarchy while views of task sequence are more likely to be conceived of as networks. Access paths should reflect the way users currently view their system, a view which should have been discussed and agreed with the users.

For instance, librarians may view a library system as a hierarchy of rooms, racks, shelves and then books for a physical layout part of the system. In contrast, for retrieval and cataloguing, the view may be one based on a current book classification system, such as Dewey or Library of Congress. But the issue desk may be conceived as a network of tasks such as checking the borrower's library ticket, recording the loan and date-stamping the book.

5.1.2 Mapping Logical to Physical Modules

To do this, interface design will use some of the principles of Structured System Design. As many system designers use these methods, their application to interface design is nothing new. For those who are unacquainted with the method, the basic idea is to divide up the system (or interface) into parts, called *modules*. The content of a module is determined by the axiom 'one module one purpose' or in interface terms 'one interface module does one, and only one, thing' (for example, a data entry screen accepts data but does not have editing operations mixed up with it; editing is done by another screen or overlay). This idea is called 'cohesion' in systems design. It aims to produce modules which carry out activities which are functionally related or in plain English 'serve one purpose'.

This notion is similar to the goal-oriented functional decomposition carried out in task analysis, therefore most tasks should show good cohesion. However, as the logical system is translated into the physical system within the constraints of available hardware and system software, it is necessary to preserve the cohesion of tasks as far as possible. The justification for introducing this criterion into interface design is identical to that employed in system design, namely cohesive modules are easier to identify, understand and maintain. An interface has to be maintained more often, probably, than other parts of the system, therefore it is important that it is easy to change. Dividing the interface into logically distinct modules makes identification of the location of change within the system easy, and minimises the spread of undesirable changes within the system. The combined effect of structured design of the program modules which implement the interface is to help system maintenance and to make the interface easier to understand for the user.

How logical interface modules map on to physical programs and sub-routines is a program design problem and will not be considered further. The interface design problem is to construct an interface which makes it easy for users to locate, understand and use its various functions. Thus although an edit screen and the data entry screen may call the same sub-routine to display a form, the two parts of the interface should be logically different to the user. This difference should be designed explicitly to prevent users being confused about which part of the system they are in.

Mapping of logical to physical modules may be 1:1, but if this is not so then defining interface modules presents the same problems as defining modules in system design: where are the boundaries and how big/small should a module be? Interface design, fortunately, has a starting point to guide these decisions in task analysis. The task sequence should be examined and the break points noted. Break points occur at the end of any series of sequentially related operational steps; in reality this means when there is a pause, for instance, at the end of a record during data entry, or when one life has been lost in a space invader game. The sequence between break points should form one cohesive sub-task which becomes one logical interface module. Close mapping of tasks to interface structures may not always be possible, especially when error and exception sequences may interrupt tasks.

Design continues by elaborating the dialogue within the modules. More actions are added to increase user control of the system and to provide support as help screens, etc. Before proceeding to detailed design, the principles upon which dialogue design is based are reviewed.

5.2 Principles of Good Design

Guidelines for good design features have been proposed by several authors in the human-computer interface literature. While no definitive set exists, there is a certain consensus and the following features are a distillation of recommendations by several authors. The principles relate only to control of the dialogue; separate guidelines are necessary for presentation of information and these are dealt with in the next chapter.

- *Feedback*: always provide users with messages to inform them what is going on, especially if there is going to be any significant delay in response time. Failure to do so leaves users wondering if they or the machine are at fault, and often causes them to press Control-C to find out what has happened
- *Status*: provide a message informing users which part of the system they are in. In large systems users may forget which facility they are using,

resulting in them issuing the right command in the wrong context. This can have unfortunate consequences

- *Escape*: allow users a method of terminating an operation and escaping from options. Many operations are selected accidentally and one of the most frustrating features of a bad interface design is being locked into an option you do not want
- *Minimal work*: try to save users' effort when operating the interface. This can be effected by using the minimal number of dialogue steps necessary (for example, do not use two question and answer steps where one will do) and by reducing the amount of typing for users with abbreviations and codes. Long-winded dialogues may be supportive at first but users quickly learn dialogue steps and slow, multi-step dialogues soon become frustrating
- *Default*: set default replies where there is a predictable answer; this again saves the user work
- *Help*: provide on-line help whenever possible. Help has two functions: first as a learning aid for users who are too lazy to read manuals, and second as an aide memoire for experienced users who need confirmation of some detailed aspect of an operation. Help should be layered or nested so the information pertains directly to the option or facility which the user wants to know about
- *Undo*: mistakes will be made and users will want to backtrack in a dialogue sequence and start again. The interface should provide the ability to go back and recover a previous state; for example, in word processing the previous version of the paragraph being edited
- *Consistency*: the format and execution of commands should be consistent throughout the interface. For instance, the escape command should use the same code (E to exit) at all levels and should have the same effect (for example, terminate the operation and return up one level in the interface hierarchy). Consistency reduces the amount users have to learn about an interface

Guidelines, however, are only useful if they are applied, but their application will often require compromises between two or more conflicting factors (for example, should feedback and acknowledgement be given at every step of the dialogue or will over-attentive messaging merely annoy the user?). Design decisions will remain human value judgements involving trade-offs between contradictory demands of a design; however, guidance can be given as to where during a dialogue guidelines should be employed to ensure the design process is methodical, if not perfect.

5.3 Putting Principles into Practice

At this stage the dialogue consists of a series of steps organised into modules which correspond to the user's task. While the basic sequence of the dialogue steps is taken from the Structured English task description, the steps within each module need organising into a coherent order and supplementing with additional steps so that the dialogue provides the correct choices for users at the correct time, gives appropriate messages and allows the user control over the interaction.

This could be done intuitively using the guidelines of good design, but improvements can be made, if not guaranteed, by planning the dialogue using network diagrams to show the interconnection between each question and answer step. The ability to trace pathways through a dialogue has two advantages. First it enables designs to be verified to ensure that bad practices are eliminated, such as answers which cause the system to crash and leave the user in limbo without a message; and secondly, guidelines can methodically incorporate good practices into a dialogue.

Interface designers have used two main methods of detailed design: dialogue specification languages and dialogue network diagrams (see chapter 4). The form of diagram and specification varies from author to author but most diagrammatic methods owe their heritage to State–Event transition diagrams. These map the progress of sequences of events within a system and have two basic components: a state which is an object or entity at rest, and an event which is something causing one state to finish and an object to change from one state into another.

Translated into dialogue terms, a state will be the computer awaiting a user's reply and there will usually be a message associated with this state either as a prompt or a feedback message relating to the last reply. The user's reply is an event which the interface has to deal with; it changes the interface from one state into another as the computer reacts to the user, issuing messages and performing actions until it requires more human interaction. In this way the whole question and answer sequence in a dialogue can be described and planned.

Dialogue network diagrams

In dialogue network diagrams, a state (or question) is represented as a circle, which is a resting state in the human–computer dialogue when the computer requires human intervention before proceeding to an event (or answer). This is represented by an arc, which shows the change between two states, each arc being dependent on the characteristics of the user's reply (such as valid data, invalid data, escape command). Each arc is annotated with the conditions which cause it to be invoked. These

conditions can then be cross-referenced to systems and program design documentation to ensure correct programming of the interface.

Diagrams read from top to bottom and concurrency can be expressed by two parallel sequences of circles and lines in one diagram, each showing, for instance, activity in separate screen windows. In some cases concurrency will need to be expressed within one sequence; for example, when a long-lasting status message is displayed it is useful to illustrate its presence throughout the dialogue. In this case a circle is used for a message state and a dotted line indicates its presence during the dialogue although there is no state-event change.

The transition between states may involve several events from the computer viewpoint, but these can safely be ignored if there are no implications for the user-system dialogue. However, if there is going to be a significant delay in response time before the computer can accept the next command, then this is a significant dialogue event which is shown as a bar on the event arc representing a delay in the dialogue due to computer processing time.

Other features illustrated are default settings of replies, and timeouts when the computer controls transition between states after a certain time period if it has not received a human reply. These are shown as an arc

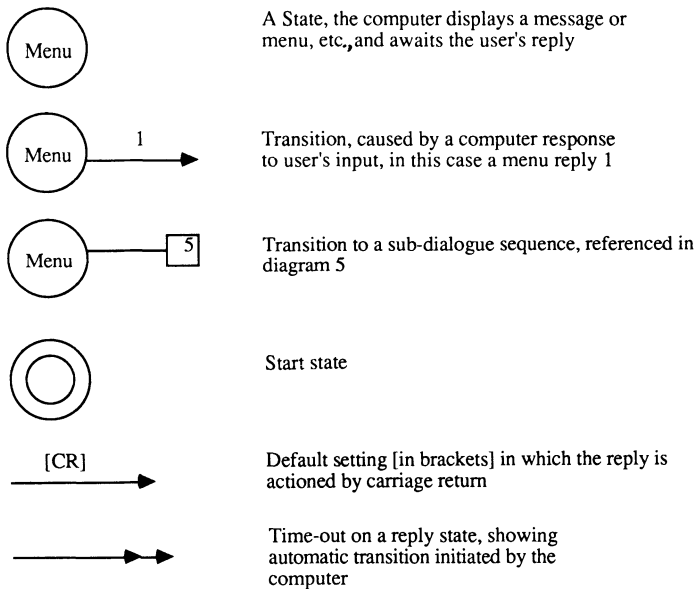


Figure 5.3 Dialogue network diagram components (adapted from Kieras and Polson, 1985).

marked with a double arrow head. Diagram components are illustrated in figure 5.3.

Network diagrams can be nested hierarchically to deal with complex sequences. A square is used to represent a call to a sub-dialogue sequence. Sub-dialogue sequences are labelled on the top-level diagram. For instance, in a command language the interpreter will be called when a command string has been entered, the parse sequence may be shown as a sub-dialogue (possibly using a different notation such as a parse tree); and in data entry, overlays may be called to deal with exception cases in an input sequence. A sample sequence is illustrated in figure 5.4.

Dialogue diagrams are not suitable for the design of complex command languages, where other techniques such as syntax graphs and grammars have to be used; but for dialogues of simple to moderate complexity, network diagrams work well.

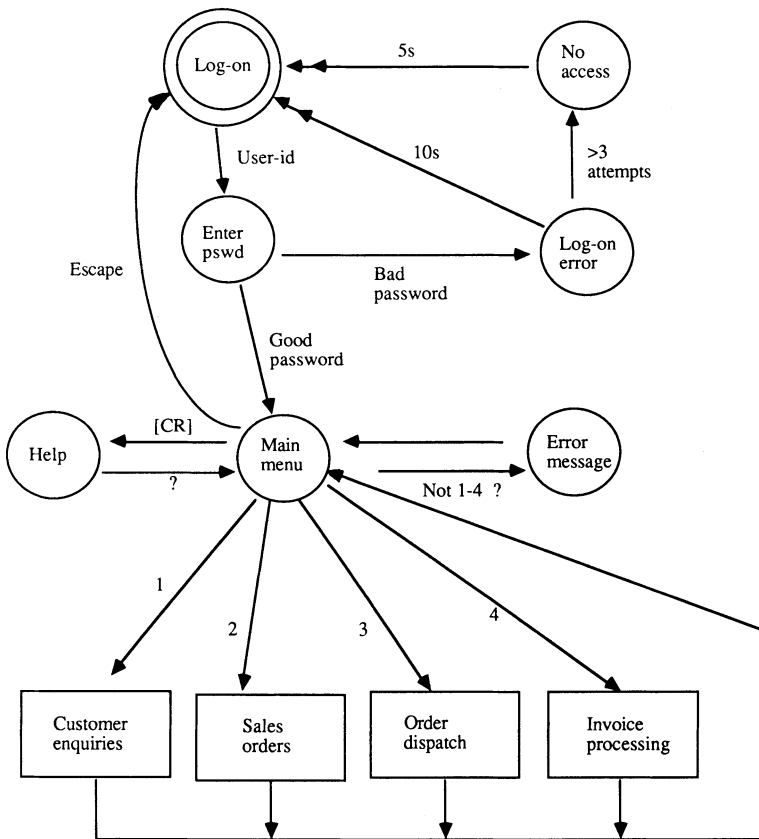


Figure 5.4 Dialogue network diagram showing the main menu in an information processing system.

5.4 Checking the Design

One of the advantages of network diagrams is that they provide a quick visual cross-check to ensure that principles of good design have been employed. This can be carried out by checking the destinations of arcs. Most circles should have at least three arcs leading from them: a normal reply, the invalid reply leading to an error message state followed by return to the previous dialogue step for re-input, and an escape route to exit from the step to a previous break point in the dialogue.

More sophisticated implementations may have five arcs from each data entry state: the three above plus a help arc which leads to a message state and waits before returning to the previous dialogue step, and an undo arc which will form a separate dialogue sub-sequence of its own.

Dialogue networks can be verified in two ways: first to ensure that the connections make sense, for example, error pathways terminate with a request for further input, escape routes take the user out of an operation at a sensible place, and second by checking that the appropriate number of arcs is present at each step to ensure that design guidelines have been adhered to. For instance, menu-selection dialogue steps can be checked to ensure an error pathway is present, an escape route is provided, all options are present and, optionally, help and backtracking facilities are present.

Network diagrams can also check the efficiency of a design. The number of dialogue steps should be examined; a large number of steps with only two branching arcs following simple questions (of the Yes/No type) should be viewed with suspicion. Such a dialogue is likely to be too long-winded for all but the naive user.

Sequences can be examined for defaults; if there are none, each step should be examined to determine whether pre-set replies could be included. Back-up information, such as status messages, should be included if not already present. By following through the dialogue sequence on a diagram, good design principles can be incorporated, although it should be remembered that good designs cannot be guaranteed, and are finally dependent on experience.

5.5 Summary

Detailed design starts by mapping the task design to interface modules employing the principle of cohesion as a guideline. The access mechanism, modelled on the user's view and choice of interface type, is added to the design to provide user control, and the overall design expressed in an interface structure diagram. Access also depends on the task structure, which may suggest a hierarchy, network or, in an unstructured domain, direct access to task fragments.

Dialogues are based on task sequences, but additions are made to incorporate good design practices for user guidance and support, such as undo, help, escape, default and feedback. Dialogue sequences are designed using network diagrams which show all the possible pathways through a dialogue as a series of state–event transitions. Nodes represent states which are computer messages and displays, arcs are transitions and annotated with the human reply which triggers the transition. Network diagrams can be verified by visual inspection to ensure that good design practices are adhered to. Escape, help and undo arcs are expected at most dialogue steps, and pathways should contain messages giving relevant feedback.

Further Reading

For details of GTNs, see Kieras and Polson (1985). Dialogue design guidelines can be found in Gaines and Shaw (1984) as a set of general ‘proverbs’; for more detail consult Smith and Mosier (1984) or Rubenstein and Hersh (1985).

6 *Presentation Design*

This chapter gives general principles and guidelines for the display of data. Presentation design for most interfaces involves screen design although other media, such as voice, will play an increasingly important role in the future. This section concentrates on the general approach for VDU screen design; detailed guidelines for different types of screen and use of graphics are given in later chapters.

Presentation design aims to display information as efficiently as possible for human perception and to structure the display in such a way as to draw attention to important items of information. Presentation design is concerned with general structuring of the display and detailed design of field formats. The following guidelines apply primarily to VDU screen displays, although most of the principles may be applied to hardcopy reports as well. The designer should be aware that reading VDU screens and printouts does differ. Procedures for screen design are described first, followed by investigation of general topics of presentation design: attention and highlighting, use of colour, messaging, abbreviations and codes, and screen layout.

6.1 Screen Design Procedure

Information can be displayed in text form or by using graphics. Text has to be displayed using characters but figures may be shown either in tabular format or qualities of the values can be expressed using graphs. Generally, the more information which can be shown in a graphical form the better, because information is assimilated more easily in picture form. However, to present information graphically requires interpretation, which may lead to users perceiving different facts from those that would be apparent from reading the raw data itself. Graphs are useful for showing trends in data and creating impressions of differences, but they are not so useful for accurate and detailed analysis of values.

The choice of whether to use graphics or character displays will be determined by the user in conjunction with advice from the analyst. A general guideline is that if figures are to be used for detailed analysis in which values are important or if data values are going to be abstracted from

the display, then character display should be used. In contrast, if overall qualities of the data need to be communicated, and values are not critical in the analysis, then graphics is a more effective medium.

To be effective, displays need to be structured. Overcrowded displays cause mistakes in reading and eyestrain. Effective presentation has to solve a dilemma of displaying the maximum amount of information on the small space of a VDU screen, while at the same time not overcrowding the screen with too much data. If too little data is displayed, the users will have to page through endless screens to find the data that they need; display too much and users have the problem of not seeing the wood for the trees.

Display design should start with design of the display structure. First information has to be grouped into blocks and the blocks ordered in a manner most useful to the users. Ordering and grouping of data will depend on the usage, and the dialogue specification may be taken as a starting point for screen design. Dialogue modules may map directly on to screens and detail within screens can be specified by inspecting dialogue network diagrams and segmenting sequences according to closure events imposed during task design. The dialogue segments can then be mapped on to screens, overlays and windows. Dialogue diagrams also give a specification of the message types that will have to be displayed.

Screen display specifications at this stage will consist of a sketch of screen areas, windows and overlays with lists of data items and messages to be displayed in each area. This specification is reviewed to structure and organise the display further. The objective of grouping data is to place data items which will be used together in the same place and make it easier for the user to find discrete data items. These two objectives may well be in conflict. The better known the data usage, the easier is the analyst's task of displaying data relevant to a task, and data items can be effectively grouped by a variety of user-defined criteria. However, when data usage requirements are ill-defined or the usage needs produce conflicting groupings, dialogues for data retrieval and dynamic configuration of displays have to be designed.

Examples of grouping by usage are placing figures for comparison together, such as planned budgets and actual spend. Groupings by category uses the identity of some object that the data belongs to, such as branch, district, regional sales figures, or some quality inherent within the data items, for instance, all counties with above average rainfall. If the usage cannot be anticipated then a compromise is to group data belonging to entities using the results of data analysis to determine display contents.

Once the contents and overall structure of the display have been decided, more detailed design is carried out to create a mock-up of the display. The display samples or prototypes are tested with users for acceptability. Early user testing of interfaces is a good way of obtaining feedback, not only on screen designs but also on the functionality of the

system itself. When users see part of the system they invariably venture opinions, whereas written specifications may be accepted without any feedback. In summary, the steps in screen design are:

- Identify system inputs and outputs. These will be part of the systems analysis documentation
- Segment the dialogue specification into screens, overlays and windows, using closure events to determine boundaries of sub-dialogues
- Identify user requirements and user characteristics. This will form part of the user analysis which determines the appropriate level of support, prompting and messaging in a screen
- Describe in detail the format of data items and messages to appear on the screen
- Design screen structure, starting with the general layout of the screen, then adding headings, titles, prompts and error messages
- Test screens with users: redesign if necessary

Systems inputs and outputs will be described in the data dictionary created during systems analysis. The important factors for screen design are to identify data flows across the system boundary, list their contents from the data dictionary and describe the screen function, that is, data entry, data display or conversational (a mixture of both). The dialogue network specification refines design to areas within a screen. Different functional pathways within a dialogue should be mapped on to screen areas reserved for their purpose, such as working, command, help and error-processing areas.

User characteristics have a bearing on the amount of support information which should be provided. Screens for naive users will require complete prompts and detailed explanatory instructions, although the amount of instructional and prompting material necessary will decrease as user expertise increases. Abbreviations and short prompts should only be used with skilled users or with novice users who may be expected to acquire skills quickly through frequent use.

Description of data items amplifies the amount of information already present in data dictionary entries by adding information needed for screen design, such as data item relationships, field size and format. A typical data item description may be:

<i>Data name</i>	<i>Size</i>	<i>Type</i>	<i>Req'd/Opt</i>	<i>Validation</i>	<i>Prompt</i>
Customer Name	30	X	R	LUT	Cust. name
Address	30	X	O	—	Address
City	20	X	O	—	City/Town
Post Code	7	X	O	—	Postcode
Vehicle type	15	X	R	LUT	Vehicle
CC	4	9	R	Range	Engine cc
Cover	1	X	R	LUT	Ins. cover

Period	2	9	R	Range	Cover. dur.
Start date	6	D	R	DD/MM/YY	Start date
Driver age	2	9	R	Range	Driver age
Years exp.	2	9	O	Range	Driver yrs
Other drivers	20	X	O	—	Ex. drivers

Data grouping in order by:

Driver details
 Policy sought
 Driver experience

Validation may be look-up table (LUT), reference list, range check, check digit etc. An extra column may be added to specify the effects of validation failure (fatal, warning) and the error message issued to the user.

Supporting data has to be added to the basic requirements for:

Screen title
 Status information: screen page, file display, current system function
 Section headings
 Messages and prompts
 Instructions and help

The basic display and supporting data are mapped, together with space to separate blocks of information, in the display area provided by the VDU hardware being used. General screen design allocates areas of the screen layout to data display/data entry, control, error messages, titles and headings. A screen sketch is prepared showing the approximate layout, as depicted in figure 6.1. The sketch is then refined using detailed guidelines to create a screen layout specification on a VDU layout chart. Design guidelines for messaging, use of colour and general formatting are necessary for detailed formatting of screens. The following sections examine these issues.

6.2 Detailed Display Design

Issues covered are highlighting important information to attract user attention, use of colour, messaging and abbreviations.

Attention and highlighting

One of the most important effects which has to be achieved when presenting information is to draw users' attention to important items. This

A M E N D P U R C H A S E O R D E R				
Requisition number	[]	Supplier's name	[]	
Order number	[]	& Address	[]	
Order date(DD/MM/YY)	[/ /]		[]	
Nominal code(6 digits)	[]	Dept-staff-name	[]	
	<u>Item number</u>	<u>Quantity</u>	<u>Product description</u>	<u>Price(x.xx)</u>
PAGE 1	[]	[]	[]	[]
	[]	[]	[]	[]
	[]	[]	[]	[]
	[]	[]	[]	[]
	[]	[]	[]	[]
Total items	[]			Total []
Narrative	[]		[]	
U)update this purchase order, R)enter amend facilities E)xit without updating Enter option:[] (ERROR MESSAGES DISPLAYED HERE)				

Figure 6.1 Screen sketch for an order entry system.

is effected by highlighting information, which must be approached with care. Overdoing highlighting can make screens tiring to read and cause physical discomfort if too many attention-seeking stimuli are used. Also, if too much information is highlighted then the user cannot possibly attend to it all, a situation which can lead to task overload and poor performance. The objective is to highlight only when strictly necessary, and even then to use the techniques judiciously so as not to overdo the overall effect.

Data can be highlighted by many different visual attributes:

- Movement (blinking or change of position)
- Brightness
- Colour
- Shape (character font, shape of symbols)
- Size (text size, increased size of symbols)
- Shading (different texture of objects)
- Surroundings (underline, borders, inverse video)

In the above list, movement is by far the most effective stimulus for gaining attention. People are very sensitive to movement as the eye has specific detectors for that purpose. After movement, size and colour are probably the most effective, but the scale of the effect depends on how the attributes

are used. Brightness is not so effective. People can only distinguish a few levels of brightness so it should be used sparingly. Over-bright images are unpleasant to read and should be avoided.

Shading is effective for drawing attention to part of the screen and does not run a high risk of presenting too strong a stimulus. Surrounding screen areas or drawing explicit boundaries can be used effectively in many ways. Text may be underlined or surrounded by a box, the background can be shaded or coloured, and in inverse video, the complete contrast to the normal image makes a very effective stimulus. Colour is a complex subject in its own right, and is reviewed in the following section. As well as using attributes of the displayed item and its immediate surroundings, highlighting and attention markers can be designed as indicators or warning icons. Care has to be taken that the user population interpret the warning icon as the designer expects.

The design should achieve a pleasant display which guides the user to important data items but does not present too many conflicting stimuli. In figure 6.2 the screen has been overloaded with attention-seeking stimuli.

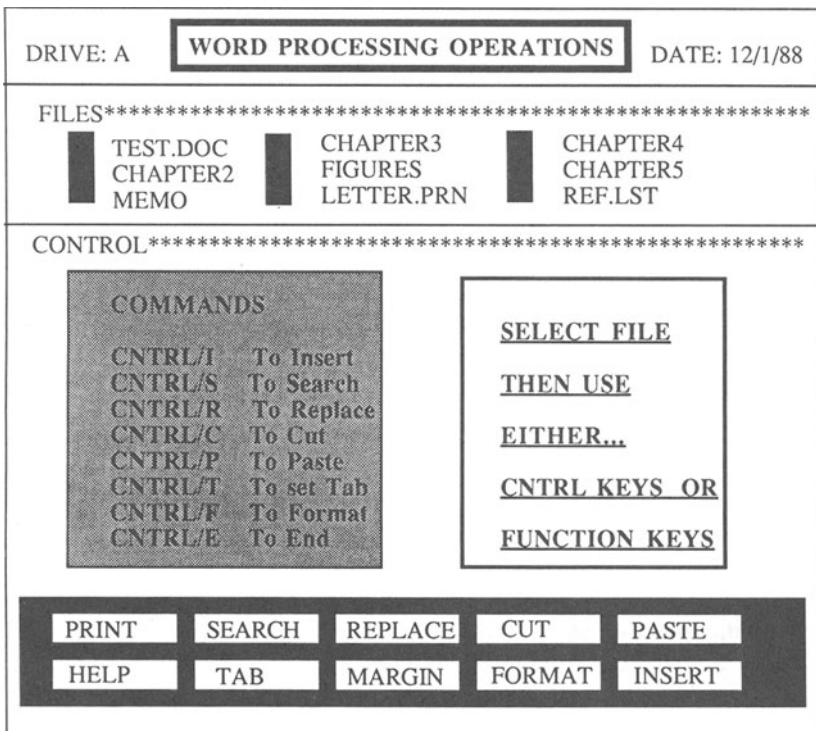


Figure 6.2 Example of too many stimuli on an overcrowded screen. The use of highlighting in several different parts of the screen results in too many strong stimuli competing for attention.

This screen would fail to achieve its objective and be unpleasant to the user. A better design is illustrated in figure 6.3 in which an attention-seeking display has been limited to the minimum necessary items.

Drive: A		WORD PROCESSING OPERATIONS		Date: 25/1/88										
Editing : FIGURES														
TEST.DOC	SALES.M3	SALES.M4												
SALES.M2	FIGURES	SALES.M5												
MEMO	LETTER.PRN	REF.LST												
Order delivery dates														
5/1/88	Order lead times													
6/1/88														
8/1/88														
11/1/88	January week													
11/1/88	1	2	3	4										
15/1/88	_____													
16/1/88														
18/1/88														
19/1/88	3	4	3	0										
20/1/88														
<table border="1"> <tr> <td>PRINT</td> <td>SEARCH</td> <td>REPLACE</td> <td>CUT</td> <td>PASTE</td> </tr> <tr> <td>HELP</td> <td>TAB</td> <td>MARGIN</td> <td>FORMAT</td> <td>INSERT</td> </tr> </table>					PRINT	SEARCH	REPLACE	CUT	PASTE	HELP	TAB	MARGIN	FORMAT	INSERT
PRINT	SEARCH	REPLACE	CUT	PASTE										
HELP	TAB	MARGIN	FORMAT	INSERT										

Figure 6.3 Example of better screen design. The use of highlighting has been restricted to the minimum necessary and the screen is less crowded.

Use of colour

Colour is a very effective technique for highlighting and may also be used for grouping information, differentiating between information, and coding simple messages (red = danger). Colour also has aesthetic qualities and properly used colour displays may appear more pleasing and restful than black and white. Visual resolution of detail is better in monochrome so there is a trade-off between the impression made by colour and the amount of detail to be displayed. Colour is a strong stimulus which it is easy to overuse.

Colour has three qualities:

- (1) *Wavelength*: this determines the basic colour spectrum from red to blue.
- (2) *Saturation*: the amount of white mixed with a colour. A low saturation colour has a lot of white in it, hence a low saturated red is a pink or rose colour.
- (3) *Brightness or hue*: this is the measure of the colour luminance.

All three qualities interact to give subject impressions which are poorly understood, and further discussion of this topic is beyond the scope of this book. Despite imperfect information on the effects of colour, some guidelines may be given:

- Limit the number of colours in one display to a maximum of 5 or 6
- Display unhighlighted information in low-saturation, low-hue colours, such as unobtrusive pale colours
- If colour is being used to code information, make sure the user understands the code

Most terminals support 6 or 7 colours (namely green, yellow, red, blue, turquoise, pink and white) and possibly shades in between to give an overall range of 16 colours. The guidelines for colour coding and use are:

- To show status: red = danger/stop, green = normal/proceed, yellow = caution
- To draw attention: white, yellow and red are the most effective
- To order data: follow the spectrum (red, orange, yellow, green, blue, violet)
- To separate data: choose colours from different parts of the spectrum (red/green, blue/yellow, any colour/white)
- To group or show similarity: use colours which are close neighbours in the spectrum (orange/yellow, blue/violet)

Note that colours have different qualities of subjective brightness and that colour affects shape resolution. Characters and detail which require good visual acuity should be displayed in yellow or white; background material is best displayed in blue which appears most restful. The common colours have visibility characteristics:

Red: low symbol luminance—poor visual acuity.

Yellow: good visibility over a wide range of luminance, best visual acuity.

Green: good visibility over intermediate range of luminance.

Blue: good visibility at low luminance, poor acuity.

A final note of caution when using colour is to remember that 9 per cent of the population is colour blind, with red/green blindness being most

common. Although colour blind people can discriminate colours using black and white shades, the designer should check that use of colour is not going to impair performance of these users.

Messages, abbreviations and codes

Presentation of text occurs in design of titles, headings, prompts, error messages and control instructions. A few simple guidelines should be adhered to whenever text is being used:

- *Keep the wording simple*: avoid computer jargon, although use of user jargon words may be necessary
- *Be concise*: do not include any words and phrase which are not strictly necessary
- *State the positive rather than the negative*
- *Use a polite but not over-familiar tone*: use of 'please' always helps but too many 'have a nice day' or 'hello I'm your friendly XX computer' messages irritate after a few days
- *Use the active voice of verbs rather than the passive voice*: for example, to Cancel order-press C, and not Orders are cancelled by pressing C

Messages should always be given in full unless the constraints of space are unavoidable, in which case abbreviations will be necessary. Codes may be necessary as a further form of abbreviation, if space or keystrokes are at a premium. Such circumstances may be found in data entry dialogues with keywords and command language dialogues.

When using abbreviations adopt a consistent approach and try to avoid exceptions to the rule. Abbreviations should be of the same length, the number of characters being a trade-off between typing time and abbreviation clarity. One approach is to abbreviate either by truncation or by compression, thereby producing a mnemonic code, that is a meaningful code in which the abbreviated word contains some clue to the identity of the whole word. Truncation removes trailing characters from a word, leaving the front few characters to convey the meaning, such as DIRectory. Generally, truncation is the easiest and most effective technique but it does run into problems of duplicates. With larger code sets, truncation becomes a less viable technique and compression has to be used.

Compression techniques aim to preserve something of the word structure while reducing the number of letters. Words are composed of syllables and one effective technique uses letters that represent the sound stressed in the syllables as the word is spoken. An example is the airline airport codes; for example, London-HeathRow becomes LHR, New York-Kennedy becomes NYK. Simpler techniques such as eliminating either vowels or consonants are not as effective; for example, compare Mchstr, Mancetr and MCR for Manchester.

Sometimes simple abbreviation will not suffice and a more complex coded representation is required. Codes are used for uniquely identifying many objects within systems and including information within the code can be helpful in the design of system processing. However, the human factors objectives in code design are to make the code as easy to understand as possible.

Basic code types are:

Hierarchical: codes in which each digit represents an object in a particular part of a classification hierarchy from super-group to group to sub-group, etc. An example is Vehicle class = 456, where 4 = Commercial vehicles, 5 = under 20 tons, 6 = Ford.

Faceted: in these codes each digit has meaning independently of the next and categorises one property of an object. For example, Manufacturer's code 9742, where 9 = mild steel, 7 = 7 mm diameter of the head, 4 = 4 cm length, 2 = product type, a screw.

Mnemonic: must be letter-based codes which contain some meaning within them.

Significant: these are derived from some measure related to the object which they describe; for example, in a matrix, 1735 refers to location row 17 column 35.

Derived: the code is produced by an algorithm which converts the original word or character into the code letter. All secret ciphers are derived codes; a simple example is letter to number conversion, 0126 representing AZ. More complex ciphers use conversion tables and mathematical conversion algorithms.

From the human factors point of view, a code structure should be made explicit to help chunking, thus for faceted and hierarchical codes:

124577659 is bad; 124-577-659 is better

In faceted and hierarchical codes the interdependencies between digit location should be kept to a minimum; the more constraints of the type 'if you have a 9 in column 1 then you can't have a 4 in column 2', the worse a code will be to use.

6.3 Summary

Presentation design takes the system input-output requirements and the dialogue specification as a starting point. Dialogue modules may map directly on to screens; however, segmentation of the dialogue according to

function may be necessary. Dialogue segments are then mapped on to screen areas, overlays and windows, depending on the target interface hardware and software.

Detailed design needs to consider use of highlighting; too many strong stimuli can create unpleasant designs. The range of attention-seeking stimuli in approximate order of potency is: movement, shape, size, colour, brightness and texture. Colour should be used sparingly and care is required when attempting to colour-code information; although as a method of improving the overall appeal of design, colour can be very effective. Messages should be concise and relevant with no jargon. Abbreviations may be necessary whenever there are constraints of display space and economy. Truncation abbreviations are usually favoured, although compression techniques can also be effective. Mnemonic codes which preserve meaning should be used whenever possible, but numeric codes may be required for processing efficiency. In this case the code should be easy to understand.

Further Reading

Galitz (1981) is one of the most comprehensive sources of screen design guidelines, but see also Huckle (1981).

7 Data Entry Interfaces

Data entry concerns input of any data items for computer processing. Data entry interfaces are the part of computer systems with which end users spend most of their time. These interfaces are also one of the most error-prone parts of computer systems and have given rise to the acronym GIGO (Garbage In Garbage Out). The design of good data entry interfaces should aim to prevent GIGO and make data entry as efficient and pleasant as possible for the user.

General data entry guidelines are described and then different types of data entry interface designs are examined.

7.1 Data Entry Guidelines

The general objectives for data entry are to save the user work, and to make entry error rates as low as possible. This is achieved by keeping users' memory load as low as possible, making the interface predictable and consistent, protecting the user from making mistakes, and automating as much of the data entry as possible. Data entry guidelines aim to give the user freedom to control entry as efficiently as possible. One method of automating data entry is to use specialist hardware reading equipment, such as Optical character recognition, Bar code readers, and Magnetic ink character recognition, as detailed in section 7.4.

Within the constraints of software design, reduction of the users' workload can be achieved by:

- Setting defaults for commonly entered items
- Using codes and abbreviations
- Automatically filling-in previously entered items, such as customer name and address, from file
- Using pointing responses and selection from a list, if entry is from a limited set of choices

Data entry screens should be designed to model the input form as closely as possible. If no input form exists or the old input form is poorly designed and difficult to use, a new screen layout will have to be designed.

Data items should be grouped together either according to their frequency of use, or their importance, or sequence of entry. The choice of

which grouping criterion to use should be made in consultation with the user. Most data entry tasks involve a transaction. Transactions in most information systems are described by a paper document—a distinct document which is created and then processed by the system (for example, customer order, hospital admittance record, export shipping document). When entering transactions, data grouping is usually by sequence of entry; however, further guidelines are given in the section on form filling, the data entry dialogue which is suitable for transactions.

Data entry dialogues should be designed to give the user positive control over the sequence of communication rather than attempting to help the user with design tricks such as automatic skip to next field, and automatic entering of default replies before the user has had time to give a command. Such dialogue features will cause frustration when they are not required, but more importantly they conflict with what people normally expect. Most people expect data entry to be like filling in a paper form by hand, in which case you have to explicitly move to the next field. Computer interfaces should conform to users' expectations even though autotabbing between fields may appear to be saving the user work.

However, autotabbing may be justified for skilled users with high transaction volumes, in which case speed and efficiency considerations are more important. The trade-off judgement illustrates how context affects the formulation of guidelines. General guidelines for data entry dialogues are as follows, but the influence of the design context should be considered when putting these into practice:

- *Explicit Enter*: validation and entry only occurs when the user presses the enter key; this allows checking within the entry for errors
- *Explicit movement*: autoskip/autotab between fields is not usually advisable, as unskilled users find the unexpected movement distracting. Use TAB or CR to move between fields
- *Explicit Cancel*: if the user interrupts an entry sequence, the data already entered, even in the current field, should not be deleted. This allows reconsideration of a cancel action which may have been a mistake
- *Explicit Delete*: make deletion an obvious action which is not easy to take without an extra confirming step—Delete Order: are you sure? (Y/N)
- *Provide feedback*: users should be able to see what they have entered. If several entries can be placed on one screen, the previous transactions should still be displayed. Feedback messages should be given to users to inform them of the next action which is expected
- *Allow editing*: editing, ideally, should be allowed within a transaction and after it has been completed; hence users should be able to edit a field that they are currently entering and to go back and change fields entered previously. A consistent method of editing should be adopted

- *Provide Undo*: allow users to backtrack to the previous 'before' state. This is often useful in edit and command sequences to correct mistaken courses of action
- *Auto format*: users should not have to enter redundant digits and characters such as leading zeros, for example 79 not 0079 to fit a PIC 9(4). Entry should not be space sensitive, for example, both A. Name and A Name should be acceptable
- *Show valid entry responses or values in prompt*: either the range or valid replies in a set should be shown, for example, enter discount value in the range of 1 to 10
- *Entry at user's pace*: users should be able to control the speed of data entry because forced work schedules will be resented.

These general guidelines are applied in specific data entry dialogues. The most common type is form filling in which data is initially captured on a paper document. Systems analysts and interface designers often have to design paper-based interaction for data entry as well as computer dialogues, as elaborated in the next section.

7.2 Forms Design

The data to be entered into computer systems may come directly from the source, which may be a person, or a measuring device, or another computer system. Alternatively, data may have already been captured on a paper document—a form. Forms design as a result tends to be an integral part of data entry design for many computer systems.

Forms play an important part in most peoples' lives and are the source of most data entered into computers. Data is entered on to forms by people using the dialogue of instructions provided by the form. This can be an error-prone process because people may mistake instructions, skip fields, give information in the wrong format, make transposition errors or write illegibly. Good form design can reduce these problems.

Data entry, whether on to a form or into a computer, is preceded by data capture. When designing data-capture procedures, the following guidelines should be considered:

- Data should be collected at source as far as possible
- Data should be entered on to the data-capture document (a form) by the originator of the data
- Avoid transcription of data from one form to another. Transcription is an error-prone process which should be avoided if possible.

Forms should be designed for ease of data collection rather than extraneous factors such as fitting into envelopes or saving on printing costs. Data collection is an expensive running cost, so if savings can be achieved by quicker, and more accurate data collection, these will far outweigh

capital costs incurred in good forms and data entry design. Forms should have a consistent design as far as possible within a system and an organisation. The more consistent designs are, the more uniform users' expectations become, and consequently their learning burden is reduced.

Forms have to be designed to capture optional as well as essential information. The design of information capture needs to identify the individuals who will fill in the form and strike a balance between having one form which tries to suit all people and many different forms with each one tailored to a particular user. The all-purpose form suffers from errors of people filling in irrelevant information and completing the wrong sections. Tailored forms, on the other hand, suffer from people having difficulty getting the right form for their needs and accidentally filling in the wrong one. How many individual data sources to target on one form is a trade-off decision. Generally one form should have one purpose and the number of alternative form types should be kept to a minimum. If there is a sizable population which can be identified as a separate data source then a specially designed form should be constructed for them. This has to be weighed against the problem of making sure the correct people get the right form.

User analysis should be carried out for forms design as with other interfaces. User characteristics can help to decide on form design for both majority and special cases, and determine the level of instructions and prompts which will be necessary.

Forms consist of three main components:

- Data entry areas
- Supporting information, and instructions
- Titles and headings

Data fields within the form need to be ordered and grouped according to frequency of use, importance, functional relatedness or sequence of use, whichever is most important for the user. Within each group, fields are ordered in sequence of entry. In transaction-related forms fields will generally be grouped in functionally related blocks; for example, in an order form: customer details, order date and delivery details, and products ordered. Data groups should be separated by clear boundaries and the complete form should not have a surface area more than 40 per cent full of data fields and printed messages. Forms more than 40 per cent full have a cluttered appearance and impair visual searching for information. The consequences of poor design resulting from overcrowding and poor structure can be seen in figure 7.1, and the effect of remedies described above are shown in figure 7.2.

Three types of form layout are most common; caption before, caption above, and caption and box designs—see figure 7.3. The caption and box design is favoured because it gives the best visual link between the caption

Name:	Arthur Brown	Application Ref	1787286CB2
Address :	12 The Avenue, Milton Keynes MK21 3RZ		
Date of birth :	12.12.50	Area code	B
Vehicle type :	Ford	Make	Escort
CC	1600	Model	GL
		Year of Manufacture	1986
Type of Cover:	C	Ins class	3
Extras:	Mary Brown	1.1.56	
	: Windscreen option		
No claims	40%	Prev Ins	GRE
Disqual	N		
Details	-		

Crowding ??

Structure ??

Figure 7.1 Illustration of poor data entry forms design.

and the data entry area, encourages readable input, and gives a more visible structure to the form.

Guidelines for general form design are:

- (a) Make selections explicit. If there are alternatives within a form of the type 'If A fill in section 1, else fill in section 2', make sure that separate sections are clearly marked and the deciding condition is stated in the positive, for example, 'If extra cover is required please complete Section B', and not 'If no extra cover is required omit section B'. The else condition should give clear navigational instructions to the next place in the form with arrows and Goto instructions as depicted in figure 7.4.

MOTOR INSURANCE POLICY APPLICATION

New Policy Date 25/1/88

Driver details

Surname : Brown Initials : A.T.

Title : Mr (Mr, Mrs, Ms, Dr, Oth) Date of birth : 12 11 50

Address : 12, Any Avenue _____]
 : Milton Keynes _____]
 : _____]

Post code : MK8 2RU

Vehicle Details

Manufacturer : Ford _____] Make : Escort _____] Model : GL _____]

Year of manufacture : 1986] Engine: 1600 (cc)

Policy Details

Policy type : CO 3P FT

Named drivers : Mary Brown _____] Date of birth 23 11 53

Options : Windscreen _____]

Figure 7.2 Illustration of better forms design. The information has structured headings, and more prompts and instructions have been added.

- (b) The effort of form filling should be kept to a minimum. Use tick boxes, circle the code, or cross out the alternative when replies come in limited sets. This makes replies neater and less effort is demanded from the user. On the whole, tick boxes are the best method because a single tick is the most economical movement.
- (c) Many forms are filled in by two or more people. Typical of these are the 'For office use only' sections on forms. Sections for different people should be clearly separated, and if any transcription from one entry to another is necessary, align related fields as closely as possible.

1. Caption before

MOTOR INSURANCE POLICY

Name -----

Initials ----- Title ----- (Mr, Mrs, Ms, Dr, Other)

Address -----

Post Code -----

2. Caption within fill-in area

MOTOR INSURANCE POLICY

Name		Initials
Address		
Postcode	Town/city	

3. Caption above

MOTOR INSURANCE POLICY

Name Initials

--	--

Address

--

Figure 7.3 Forms design showing three types of prompt and fill-in layout.

Form layout

Slightly more printed information can be put on a form than a VDU screen, so more use can be made of delimiters to break up the form into distinct areas. Placing groups of information into boxes is an effective technique, and background colour can also be used to differentiate information. Fill-in areas should be lightly coloured, while more stimulating

MAJIC WIDGET COMPANY

SALES ORDER

▼

Customer code

Order Date

D D M M Y Y

Order type (P pro forma, N normal terms) Shipping details (H home E export)

Payment details (CH cheque CA cash) Export

Required delivery date

Country agent

Shipping agent

CCT class

▼

Product Code	Quantity	Unit Cost	Total Cost

Continuation sheet

Figure 7.4 Forms design illustrating use of tick boxes and navigational instructions.

colours should be reserved for titles and instructions. When designing more detailed layout it is important to bear in mind the following points:

- (a) Captions and prompts should either precede the fill-in area or be left-justified above the box
- (b) Data entry fields should be aligned left-justified, and if possible with a justified right margin. However, as entry fields are invariably of different lengths, right-justified margins are difficult to attain; so to create a more balanced design, it is better to aim for one row with one answer if space permits.
- (c) The filling-in area will depend on whether hand writing or typewriter completion is anticipated. For hand writing, allow $\frac{1}{4}$ -inch width per

character with extra space for separation, and a height of $\frac{1}{3}$ inch. For typewriters this can be reduced to $\frac{1}{10}$ inch by $\frac{1}{6}$ inch. Separating the fill-in area into character boxes can help legibility but runs into problems when replies over-run the number of boxes printed on the form.

- (d) When the number of characters in a reply is known, the data entry area should be subdivided to format the reply field. Character delimiters should not be too obvious or they will interfere with the reading process and make filling in slower.
- (e) If units of measure are being requested, the unit should be specified on the left-hand side of the fill-in box unless some multiple is being requested; for example, if thousands, trailing 000s are used.
- (f) Highlighting should be used for titles, mandatory fields, important prompts and instructions for filling in.

Prompts, titles and instructions

Wording on forms is vital to success. Three rules apply to all wording:

- Keep it simple
- Be explicit
- Exclude anything not directly relevant

Titles must describe concisely the purpose of a form and should be centred at the top of a document.

Completion instructions must be clear, brief and use easily understood words, especially for public use forms. Brief instructions may be located before the entry field to which they pertain, while more complex instructions should be placed at the top or bottom of the form. However, if instructions cover 50 per cent or more of the form's area then a separate instruction sheet should be used. Such detailed instructions should be put on a separate page with the order of instructions kept in pace with questions on the form. Once the form has been designed it can be used for the basis of data entry screen design, but further design is necessary for dialogue control when using form filling as a data entry interface.

7.3 Form-filling Interfaces

Form-filling dialogues are the most common data entry interface for information systems. The principal aim is to model the computer interface on the data entry document as far as possible; thus the user is familiar with the interface layout and transcription from paper to computer follows a sequence that the user knows.

Many of the guidelines for forms design also apply to the design of form-filling screens, but form-filling interfaces are a true human-computer

dialogue and the dynamic part of error messaging, data validation and computer control are extra features not found in forms design.

Data validation

Data entry is notoriously error prone. Errors may be caused by omission of a field, incorrect data being entered in a field, and number/letter transpositions. Data validation attempts to check that all mandatory fields are filled in, and that the data entered is correct, or at least reasonable. Some commonly used methods for data validation are as follows:

- *Lists, look-up tables and reference files*; all synonyms for checking data entry values against a list of all possible valid values held on the computer. The most common type of validation used is checking against a set of values, such as Customer numbers, Account codes, Part types, etc.
- *Type check, picture check*: simple check that the data is of the correct basic type, that is, numeric data was entered when expected
- *Sub-range*: the value entered is compared against a range of expected values. This is similar to list checks but simpler, for example, number check replies within the 0-99, character check A-Z
- *Check digit*: useful for numeric codes when the input values are known and faster than list checking when the code set is large. The idea is to use an extra digit which is added to the code number, having been calculated from the code itself:

Reference number	1	2	0	3	4	
Weighted by	6	5	4	3	2	
Product	6	10	0	9	8	
Sum of products	33					
Divided by prime number 11 = 3, remainder 0						
Number plus check digit	1	2	0	3	4	0

When the code number is re-entered it is checked by recalculating the check digit. The above example uses the modulus 11 technique; there are numerous other hashing algorithms for this purpose

- *Comparison check*: a comparison check assesses the reasonableness of one value by comparing it with another related value. A typical example is height/weight ratio for people. These measures follow an approximate relationship, hence if the weight was entered as 70 kg and a height as 1 m 20 cm, it is a reasonable guess that something is wrong unless, exceptionally, there is a very fat dwarf in the sample
- *Probability check*: this adds more sophistication to the reasonableness check by setting limits in the form of a range around a norm. Thus in

the weight sample a deviation of 20 per cent from the norm would be flagged with a warning error. More accuracy can be attained by using a statistically calculated standard deviation

Validation errors may be classified into three categories and different error correction actions specified for each.

- *Fatal errors*: errors which make a nonsense of further processing, such as invalid account codes, customer names. In this case the user must either re-enter a correct value or abort the entry; no other action must be allowed
- *Warning*: errors which are caused by highly unlikely values. Processing should be halted, and the user invited to re-input. However, an over-ride should be given so that the user can input the original value which may be the exception to the reasonability rule
- *Advisory*: errors which are caused by unlikely values. Processing may not necessarily be halted but a warning message should be given so the user can halt either immediately or at the end of the transaction to check and possibly edit the data

Validation messages are placed in a consistent part of the screen reserved for error control. This leads into the question of screen design for form-filling interfaces. General design was covered in chapter 6; considerations specific to data entry are now examined further.

Screen design

The screen area should be partitioned into data entry, command and error-processing areas as illustrated in figure 7.5. Alignment of data entry fields with error messages is desirable but this may not be possible if an over-riding priority is to make the working area resemble the source document which consequently fills the whole screen area.

Guidelines and a procedure for grouping information and formatting the screen were given in chapter 6 (section 6.1). A complication of many VDU terminals is the capability for local data entry and limited validation in intelligent terminals. The terminal and not the application program performs simple checks such as numbers entered into numeric fields. These features are found in IBM 327X terminals and similar products from other manufacturers. The major difference is that data is only entered into the main computer, and hence the application program, when the ENTER key is pressed at the end of a screen rather than after a CR per field. As a result a whole screenful of information is validated at once, and error messages have to be linked to appropriate fields.

This linking can be achieved by highlighting incorrect fields and constructing an error-recovery dialogue for the user to step through errors one at a time, or linking fields to error messages with a code. The advantage of 3270-type terminals is the range of screen design features not available on

15/12/87	<u>ACCOUNTS CONTROL</u>	15:23
Errors - Ready	Transaction entry	
Monthly expenses Enter Account Code <00435> Expenses: travel Cost Centre Code < 145> Date of Expenditure <12/11/87> Enter Amount £ <__ 8 9 : 5 0> Receipt Checked (Y) < Y>		Error 4 digit code req. Error current month only - Dec
Press ENTER to edit S and ENTER to save and exit ESCAPE to exit and abandon		
TAB to move to next field		

Figure 7.5 Screen layout on intelligent terminals (IBM 327X variety). Error messages are aligned to entry fields because validation occurs only after the whole record has been entered.

ordinary VDUs. Fields can be located anywhere on the screen and properties associated with fields by 'attribute bytes', which tell the host program what the field type is and control simple terminal operations on the field, for example:

- masked field—no display for passwords
- display only field
- data entry and edit field
- message field
- simple type checking

Other display qualities can be coded in attribute bytes such as highlighting, colour and inverse video. Many of these display properties are used to improve messaging, a vital component of user-system communication, only too often neglected.

Messaging

Messaging is important for conveying the type of errors and for giving users instructions to control the input sequence and error correction. As with all computer messages, the wording should be clear, simple, concise and relevant.

Error messages have often been one of the most user-vicious parts of interfaces. There is no excuse for 'Syntax error' or 'Invalid field'. Error messages must be informative, jargon free and attempt to tell the user not only what is wrong but also why, with an explanation of the correct course of action to put it right. For example:

```
Start date      : 1/10/86
Maturity date   : 12/9/86
```

Error: Maturity date before Start date: Please re-enter either date.

Messaging in prompts should be positive and active voice. In other words express the Dos rather than the Don'ts and use the active voice 'press Return to Continue' rather than the passive voice 'This sequence may be terminated by pressing the Break key'.

Handling edits and errors is one of the most complicated parts of data entry interfaces, which often necessitates creating new dialogue sequences for these tasks. Dialogue control for data entry aims to prevent errors happening, and when they do to make correction a simple matter for the user.

Data entry dialogue control

Part of the dialogue will be specified in the task design, but data entry dialogues invariably require elaboration to deal with editing and correcting errors. The dialogue should be planned with break points within the sequence to allow closure events: rest and reset points for attention. Break points for closure become more important the longer a sequence is. In a short transaction with 5-7 entries, a break after each record may be permissible, although longer transactions will need break points within a record sequence. The break points should be planned to match the blocks of information and grouping on the screen layout.

Data entry invariably requires data editing. The pace of data entry and editing should be determined by the user. Data editing screens should allow the user to check entered fields to ensure that errors not trapped by validation are found and corrected, as well as guiding the user to correct those errors detected by validation. There are several different methods of implementing editing, for instance:

(a) Prompt errored field and re-enter, for example:

Delivery date error: month out of range : 12/13/86
 Please enter in DD/MM/YY format : —/—/—

This method can be used effectively with overlays when a long sequence of data is being entered. The incorrect entry should always be shown to prompt the user. The main disadvantage is that the prompt for re-entry may obscure data already on the screen so that not all the data on the screen is visible for checking.

(b) Address errored field to re-enter, for example:

1 Customer number : 13045
 2 Customer name : J. Smith
 3 Customer address : Sunnydale Av, Milton Keynes
 4 Vehicle type : Ford
 5 Policy period : 18
 6 Policy type : C

Errors: 5 Policy period too long—12 months max.

6 Unknown policy type—valid reply codes CMP 3RD
 3FT TMP

Type field number to edit
 (1–6 or 0 to escape)

This method may be useful in long sequences of entry fields, especially when errors are not detected immediately as with 327X terminals.

(c) Edit/skip correct fields, for example:

Customer name: J. Smith
 Address: The Willows
 Address: Sunnydale Avenue
 Address: Milton Keynes
 Vehicle Type: Ford Make: Escort CC: 1300
 Policy: C Durn: 12 mths

Press Tab to skip to next field
 or Enter to save

Edit/skip field editors are quick to use and display the whole of the entry for checking as well as error messages from validation. The user moves between fields using the TAB or CR key which is easy to remember, and then types over the incorrect data. The disadvantage

with this technique is tabbing past the errored field by mistake and the tabbing time taken in long entry sequences.

While form filling is probably the most popular of current data entry interface techniques, it is time-consuming to operate and inappropriate if the set of replies is limited and predictable. In these cases other entry techniques can be applied.

7.4 Alternative Data Entry Techniques

These fall into two groups; other software designs, using menus and keywords; and hardware techniques which automate all or part of the data entry task.

Entry by menu selection

If data entry involves selecting items from a fixed list of alternatives, menu techniques can be used. The principle is simple. All valid choices are displayed on a screen and the user is invited to select one or more by entering a code number displayed alongside the item, as illustrated in figure 7.6. More sophisticated designs use picking techniques with light pens or mouse devices for users to select items displayed either as text menus or as icons.

The main design consideration for picking displays is to group items together in a logical scheme to guide the user towards the item required.

Keyword data entry

Keywords can be used as an alternative to menus when a quicker, more efficient dialogue is required. Keyword codes have the advantage of selecting an item directly by its identifier, whereas with menus, users may have to page through several layers of access hierarchy in large systems. Keywords are more flexible than menus and may be entered in different sequences, allowing for more complex transactions to be input. Keyword codes are suitable for skilled users when the data entry set is restricted. A typical example is airport codes in airline reservation systems. Keyword codes identify the starting place and destination required as mnemonics such as LGW, LHR for London GatWick, HeathRow.

Optical mark/recognition (OMR)

Optical marks can be used on forms as an extension of the tick box method. The user marks an area of the form which is then passed under

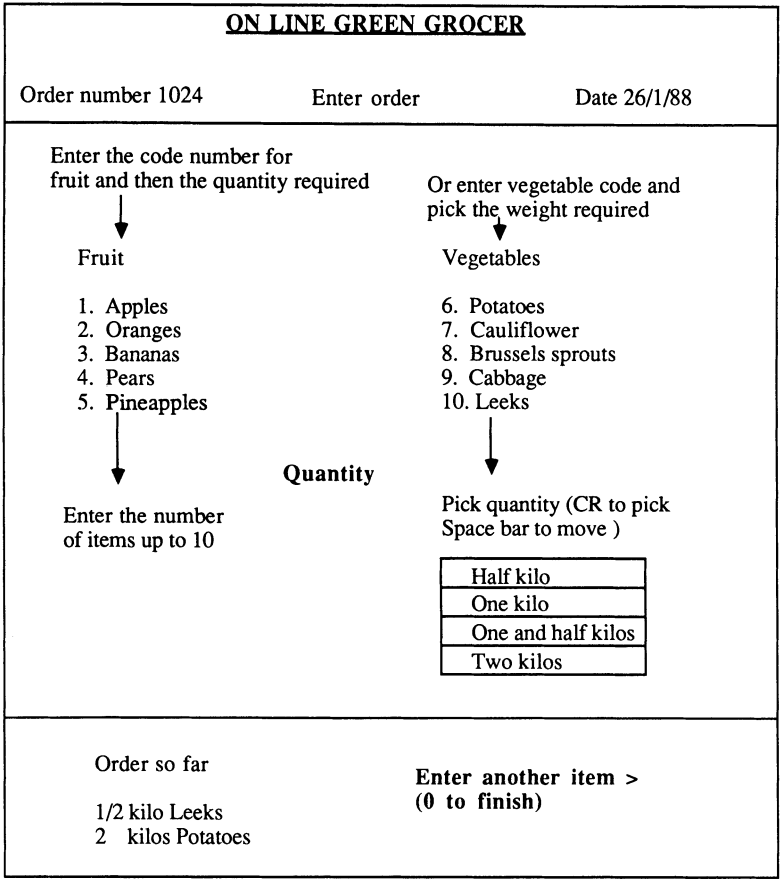


Figure 7.6 Using menus for data entry.

light-sensitive reading equipment which interprets a dark mark as yes and light marks (that is, unmarked) as no. Data suitable for entry by menu techniques can be used with OMR. The main advantages of OMR are that the source document can be used as the data entry document and entry is quick. Against this must be weighed the cost of equipment and the problems of errors due to smudges on the form. OMR techniques are useful when data entry volumes are high and direct access to computer terminals cannot be provided.

Bar codes

These are a special case of optical marks in which goods are labelled with a unique combination of vertical stripes which code a number by the

presence or absence of bars in certain positions. Bar codes are now a ubiquitous feature of supermarket packaging. The code is read by a special light-sensitive wand or bar code reader which picks up the dark bands as it is traversed across the coded area and translates the sequence of bars into a code according to the presence or absence of a dark band at position x , $x+1$, etc. The computer compares the bar code sequence against a look-up table and computes the number of the stock item.

Bar codes are a good example of considerable investment by computer manufacturers on behalf of the users (the supermarkets) to help them automate data entry. This form of hardware-dependent data entry is expensive.

Magnetic ink character recognition (MICR)

This is one of the first techniques introduced to speed up data entry. MICR printing is familiar as the odd-shaped characters used for account numbers and sort codes on bank cheques. The computer-readable bit is encoded in magnetic material inside the number and has little to do with its shape. MICR recognition requires a specialised magnetic reader which is sensitive to the pattern of magnetic code within the print.

Optical character recognition (OCR)

Computers have trouble reading printed text because, somewhat surprisingly, it is very variable. An attempt to cut down the variability was the introduction of standardised computer readable codes by the European Computer Manufacturers Association (ECMA). This stylised print enabled computers to read characters by pattern matching; however, this approach was of limited use because of the expensive printing requirement for the ECMA character set.

More recently, OCR systems have been able to deal with printed text in a number of different fonts and sizes by making character recognition systems learn the characteristics of a type face. After a few trials the computer system learns the rules for a typeface and incorporates the rules in its pattern-matching algorithms. Machines such as the Kurzweil reader can read ordinary books and newspapers more quickly than humans can. OCR equipment is expensive but costs are falling and applications involving large volumes of text are good candidates for automation, hence OCR systems have an obvious application in libraries and archives. OCR of handwriting remains a problem; some systems can recognise capitals but continuous script still defeats most machines, probably because human handwriting is variable and, only too often, illegible.

Voice data entry

Voice is still in its infancy as a data entry method but it has some appealing advantages. It is quick and can be used in environments where paper is inconvenient, such as on the shop floor where a keyboard would be unsuitable. No transcription is required and the users' own communication medium is employed. Voice data entry encompasses all the problems of voice and natural language dialogues (see chapter 9, section 9.8). At the present state of the art, limited data entry of single keywords is possible with vocabularies of 200–3000 words. Talking typewriters will soon be marketed with vocabularies of 8000 words, which is approximately the size of the human everyday spoken vocabulary.

7.5 Summary

Data entry interface design should aim to make tasks easy for the user and to minimise the input workload. The user should be in control of data entry sequences and actions should be made explicit. Forms design is important for data-capture documents. Logical layout and formatting are the most important factors of forms design, although clear prompts and instructions are also important.

Data entry can be achieved using a number of dialogue design techniques. The most common interface style is form filling, which mimics the paper operation. Form-filling dialogues are useful for complex and open-end data in which the reply set cannot be predicted. When the reply set is better known, picking menus or keyword command input may be used. Where possible, data entry should be automated. This can be achieved using optical character recognition, bar codes, magnetic characters or voice. All these methods are in limited use at present; however, voice in particular may increase for remote data entry and in environments where keyboards may be inappropriate.

Further Reading

See general references.

8 Data-Display and Data-Retrieval Interfaces

Data-display interfaces consist of query screens, file browsers, display graphics and reports. Guidelines are given for composition and layout of displays, followed by advice on more detailed formatting of data. Data-retrieval dialogues form an integral part of data-display interfaces, consequently a section is included on this topic. Graphical displays merit a section on their own; this deals with graphic display design, shape and colour, concentrating on business graphics. The chapter concludes by considering report design.

8.1 Data-display Guidelines

The inputs to display design come from the information display requirements and analysis of the users' knowledge about displays and documents in the current system. If there is an existing document which the user is likely to expect to see as part of the system, such as paper reports and summaries, then the computer display should follow the document layout if possible.

Display design has to resolve what data to display and then how much information to place on a screen. Display too little and users have to page through many screens to find the data they need, display too much and users cannot see the wood for the trees. The general aim is to display information which is appropriate for the user's task without overcrowding the screen.

To decide what to display, the following guidelines may be employed:

- Display only necessary data. Anything which is not directly related to the users' requirements should be omitted
- Data which is to be used together should be displayed together
- The data on display should be related to the task that the user performs with the data
- The quantity of data per screen, including titles, headings, etc., should not cover more than 30 per cent of the total area

Using these criteria and the users' requirements, the next step is to divide data into groups and then to structure items within a data group.

The objective is to make the data as easy to use as possible. People make sense of data by imposing structure on it; if the designer can anticipate this step it should save the user work. Users have two main problems with displays: first finding relevant data, and then finding two or more related data items. Interface design can help finding data by providing a well ordered structure for displays and by placing related data together, although the latter aim depends on establishing the anticipated usage exactly, which may not always be possible. To help wayfinding through data, the following methods may be used to structure displays:

- (a) Group data in a logical manner. This will usually be data relating to the same object (for example, a customer order), or grouping items which share the same attributes (all orders processed this month). Grouping can be by frequency of use, sequence of operations or function according to the users' views.
- (b) Order data according to criteria which are meaningful to the user. Key fields and identifiers should be placed at the top left-hand side of displays; other data may be ordered by importance, frequency of use, sequence of normal usage, mandatory then discretionary items, etc.
- (c) Structure data within lists. Sort items by one or more keys, group items belonging to the same class.
- (d) Show abstract qualities of the data if required, and use graphics to illustrate those qualities (trends, associations, differences).

The display is designed in groups of related information which are controlled by the user-system dialogue. Depending on the overall screen size, each screen may consist of a few or several sub-sections, each containing different information, unless overlay techniques have been employed. Inclusion of too many different sections impairs visual searching and locating data, as multi-purpose screens become too complex for users to assimilate. Screens should have only a few sub-sections of data, with each section separated from the next by spaces. Use of delimiters, such as &&&& ***** \$\$\$\$\$\$, should be avoided as these only increase screen crowding and add no extra information.

While screen layout is being planned, there are other general display guidelines which should be considered:

- Codes and abbreviations should be kept to a minimum. Data displays should be immediately comprehensible to the reader without having to translate codes
- If several displays are being planned, try to establish a consistent format. If users know where to expect information and how it will be presented, they have less to learn
- Provide clear headings, titles and other wayfinding information to help user navigation within and between displays

- Use the user's conventions. Follow any user model of the data discovered during analysis and keep to the user's terminology. A typical example is use of either UK or US date conventions, which are DD/MM/YY and MM/DD/YY format respectively
- Highlight important data with colour, text size, underlining or by a different font

After the display structure has been designed, detailed design depends on whether graphics or character displays are being used.

8.2 Character Data Displays

The presentation problems of character data displays are how to lay out screens and format the data items so that they are easy to find and pleasant to read. Displays may be either pure text, or tables and lists, and more frequently a mixture of both.

Character size is under software control in many displays, and while the default character size may be suitable for ordinary usage, large character sizes should be used for projection and if the reader is more than 0.5 metre away from the screen. Displays for projection as overheads should use 18–24 point characters while 10–12 point serves for normal work.

Pure text displays

Continuous capitals for text should be avoided because reading rates for capitals are slower than those for mixed text. Capitalisation should be used as in printed text and occasionally for emphasis.

Text in English should be left-justified and the right margin may be ragged as this does not impair readability. If both right and left margins are justified, equal spacing between words is preferred as unequal odd-shaped gaps distract the eye precisely because they are unequal.

Lists and tables

Numeric lists should be presented down rather than across, principally because this helps addition of totals, and because most people expect to scan a list going down rather than across.

Captions should be placed above columns:

<i>Branch</i>	<i>Total sales</i>	<i>New accounts</i>	<i>Major accounts</i>	<i>Losses</i>
London	31,234	23	123	12
East Anglia	12,124	4	65	5
East Midlands	13,433	12	59	3

Leading captions and prompts should be placed before the data and separated by a space or delimiter:

City: Manchester

Population: 1,546,000

Data fields should be left-justified for text, and either justified on the decimal point for real numbers, or right-justified for integers:

<i>Compiler</i>	<i>System time (minutes)</i>	<i>Number of users</i>
COBOL	161.68	123
FORTRAN	23.1	12
APL	54.56	21
RPG III	0.75	1

Displays should not have one static format; often the contents of a display need to be under user control, consequently a display-control dialogue is required. Simple dialogues provide users with access to a set of pre-designed displays; more flexible dialogues support user-control of the display by browsing and data retrieval.

Controlling displays

Users should be given a flexible means of accessing different displays. Some users may want to browse through a large amount of data while others need to find detailed items quickly. Without going into the level of control necessary for data retrieval, data-display dialogues should allow the user to page and scroll display screens. In page control, part of the previous display should still be visible at the top/bottom of the new page to provide the user with continuity. When using scrolling the speed should be under user control so that unwanted data can be skipped with a fast scroll and more interesting data can be inspected with a slow scroll.

In more structured databases, complex access mechanisms can be provided to control not only the display sequence but also the information content of displays by data retrieval dialogues, as described in the next section.

8.3 Data-query/Data-retrieval Displays

These displays give users more choice in what data is to be displayed. Simple data retrieval is by access to a pre-determined frame of data such as a Prestel page. Access can be provided by a menu system or by a direct

address, for example, the page address in Prestel. Frames of data or objects in a database can also be represented visually as icons. This approach was used in the Spatial Data Management System developed at MIT in which the interface was organised as a hierarchical series of iconic menus. By using joystick controls, the user could fly over the icon screens pointing at objects and navigate through a three-dimensional data space as illustrated in figure 8.1.

Most data-retrieval systems, however, aim to provide the user with choice about what is to be displayed from a database. To achieve this, data-retrieval command languages have been developed to formulate queries. These command languages are English-like, but the user has to learn a syntax and identifiers for data entities and their attributes. The usual form is as follows:

Search Entity with Attributes = X and Display Attributes, X Y and Z

A typical query in IBM's SQL query language is illustrated in figure 8.2. The basic syntax is to Select (variables/attribute values) from a set where (conditions).

People often have difficulty using data-retrieval languages. Most problems stem from poorly designed syntax and confusion about logical operators, such as AND, OR, >, <=, etc. Many users confuse logical quantifiers such as Greater Than with Greater Than or Equal To; also, compound conditions cause further problems with AND and OR conditions being mixed up because English does not distinguish the exclusive-OR from OR but may be Both conditions. Most databases have a built-in query language which is beyond the interface designer's influence; however, if a data-retrieval dialogue is being designed, the following points should be considered:

- (a) Users have to find the entities in a database which they can ask questions about. They will have to remember the names of entities, which therefore should be concise and descriptive and not terse and obscure.
- (b) Users will also have to remember the attributes of entities if they wish to select the values of attributes. Attribute names should be clear and meaningful, and attribute lists should be displayed on pop-up menus or help screens.
- (c) The syntax of a query language should follow the model of English as far as possible, because people will naturally formulate queries in a linguistic manner. Thus the query command should be like an imperative English sentence: verb, object, qualifying clause, for example:

FIND CUSTOMERS WITH ACCOUNT >10,000

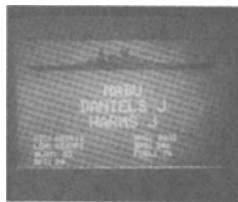
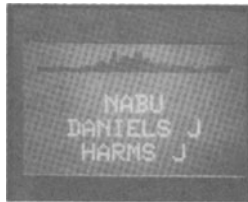
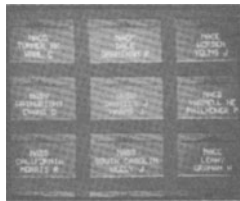
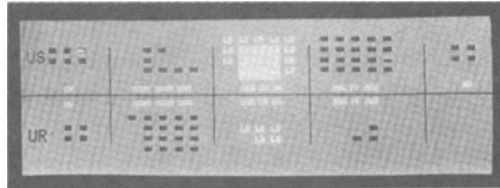
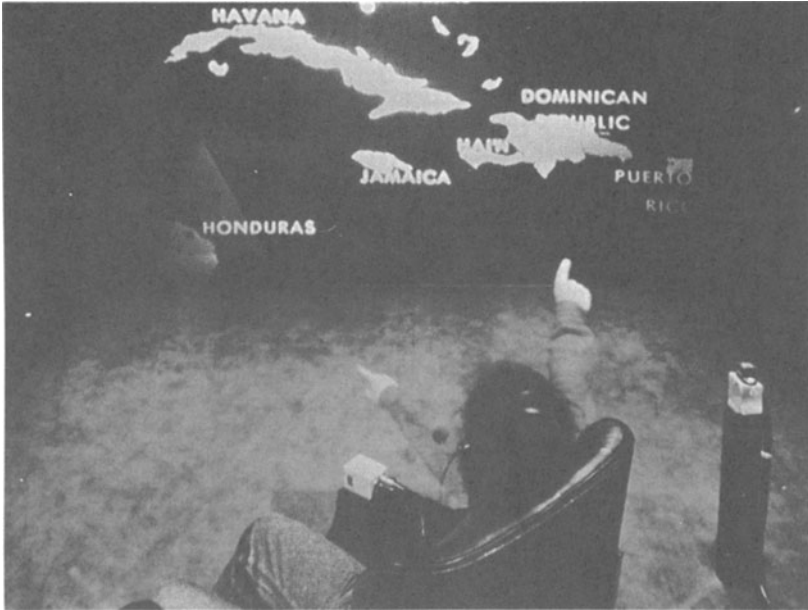


Figure 8.1 Representation of a database by a combination of icons and spatial position in the Spatial Data Management System [Source of bottom half of figure: US Naval Dept].

```

SELECT TITLE
FROM BOOKREFS
WHERE $$ INPUT
      (SELECT $$
       FROM PUBLIST
       WHERE T$ INPUT
        (SELECT T$
         FROM AUTH
         WHERE NAME = "Jones"))

```

Result

Software development: A rigorous approach
Jones C.B.

Practical systems analysis
Jones A.N.O.

The basic syntax is Select (variable/attribute name) from (entities/relations/sets) where (boolean expression). SQL can use nested syntax to express successive selections from entity sets, in this case titles, publishers, and authors. Matching conditions for attributes are specified in WHERE clauses.

Figure 8.2 Example of a data retrieval query in SQL (Structured Query Language). In English the query is 'Find the titles for books in the publication list where the author's name is Jones'.

- (d) Many searches proceed in steps as the user selects a set of likely records and then chooses from among the first set. An output file should be provided with search results, and this file should then be the input file for the next search. This method is better than complex nested search syntax because it reduces errors due to incorrect syntax and the mistyping of long command strings.
- (e) Logical operators should be clearly specified on help screens with examples of their impact. Clarifying sentences in English may be given after a query has been formulated as user feedback, for example:

This search will find customers with accounts over £20,000 and customers with accounts equal to £20,000.

- (f) Finally, data retrieval is a task suitable for decomposition into a logical sequence. Most data retrieval involves three or four steps:
- Finding which parts (or entities) of the database to query
 - Formulating the query logic and syntax
 - Refining the search if it is iterative
 - Formatting the results for printing or display

Data retrieval should make the task steps explicit and provide support for each step in turn. Too many data-retrieval languages try to do everything at once, which may benefit the expert but is of little use to the vast majority of inexperienced users.

8.4 Graphical Displays

Graphics are effective because they abstract qualities from a set of data and present information in a more 'chunked form'. But that process of analysis can introduce bias into interpretation of data and designers should exercise care when choosing graph types and in the design of layouts.

The choice of graph type is limited, to an extent, by the type of data. Data sets for graphics come in three basic forms. Values can be:

Ordinal:	boolean, that is present/absent
Nominal:	integers
Decimal:	reals

Data sets can be categorised according to the type of plot, which may be derived from a particular data value:

- (1) *Grid data*: measured values on one axis are plotted against fixed intervals on the other. It is used to show the number of members per category, or a measure per object/population on the fixed axis, such as rainfall per month or numbers of cars by type.
- (2) *Named data*: measures of the number of data items making up a set, such as government expenditure by sector.
- (3) *Point data*: measures with values for the x and y coordinates for each object. Point data may be decimal or integer for both coordinates, or decimal on one axis only.

The first two categories have a single value for each object in a measurement population; point data has two values related to an individual object, such as the height and weight of a person. Three values per object necessitate x , y , z axes in a three-dimensional plot. The other design consideration when choosing graphs is the type of analysis which is required. Users may want to show a particular quality of the data for demonstration purposes, for instance, trends, grouping, differences. Without going into complexities of statistical analysis (see Siegel, 1956 for details), the more simple treatments of data which are usually encountered in information systems are:

- (a) *Association*: the graph is to show how two measures or classes of

- objects are related in absolute terms (same value) or co-vary in some manner.
- (b) *Difference*: the antithesis of association; here the aim is to show how items differ by an absolute magnitude or show opposite patterns in variation.
 - (c) *Exception*: this is a special case of difference related to a set of items. The objective is to show the 'odd man out' in a set.
 - (d) *Trend*: aims to show a pattern in a set of values over a range, usually an increasing or decreasing trend.
 - (e) *Grouping*: this is a special case of association which aims to show relationships between many objects in a population. The inverse effect to clustering is a measure of scatter. An example is the clustering of weight and height measures around the average values of 70 kg and 1.7 m.
 - (f) *Distribution about a norm*: the graph is to show how a group of data items are spread around the average value for the population. Normal distributions are balanced with an equal number of items above and below average, with most values clustering around the average. Other distributions may show skew, more points above or below the average, or kurtosis (more points spread away from the average than expected by statistical definition of a normal distribution).

Four graph types are available in most commercially available graphics packages.

Histograms. These are also called bar charts. Histograms are suitable for ordinal and nominal data, and give a good impression of difference, exceptions and possibly trends for fairly crude measures. However, histograms waste the accuracy of decimal data and are poor at showing complex trends and small differences in measures. Histograms are suitable for grid type data, when the plot has values on one axis at fixed intervals for a measure on the other, for example, rainfall per month or as named measures (numbers of cars sold, Ford, BL, etc). The values in integers or reals are plotted on the y axis.

Pie charts. Pie charts are suitable for showing exceptions. The data items have to be members of a set which becomes the pie. This technique is effective for displaying comparisons and has a high visual impact which can be enhanced with pie and slice design, as shown in figure 8.3. Segments correspond to the value of a measure; the first segment should start at 12 o'clock and the pie is read clockwise from that point.

Line graphs. Line graphs are suitable for decimal data. The extra resolution of decimal measures is shown better in line graphs, although designers should still be aware of distortion. For instance, stretching the x axis can reduce the visual impact of difference between values on the y axis. Line

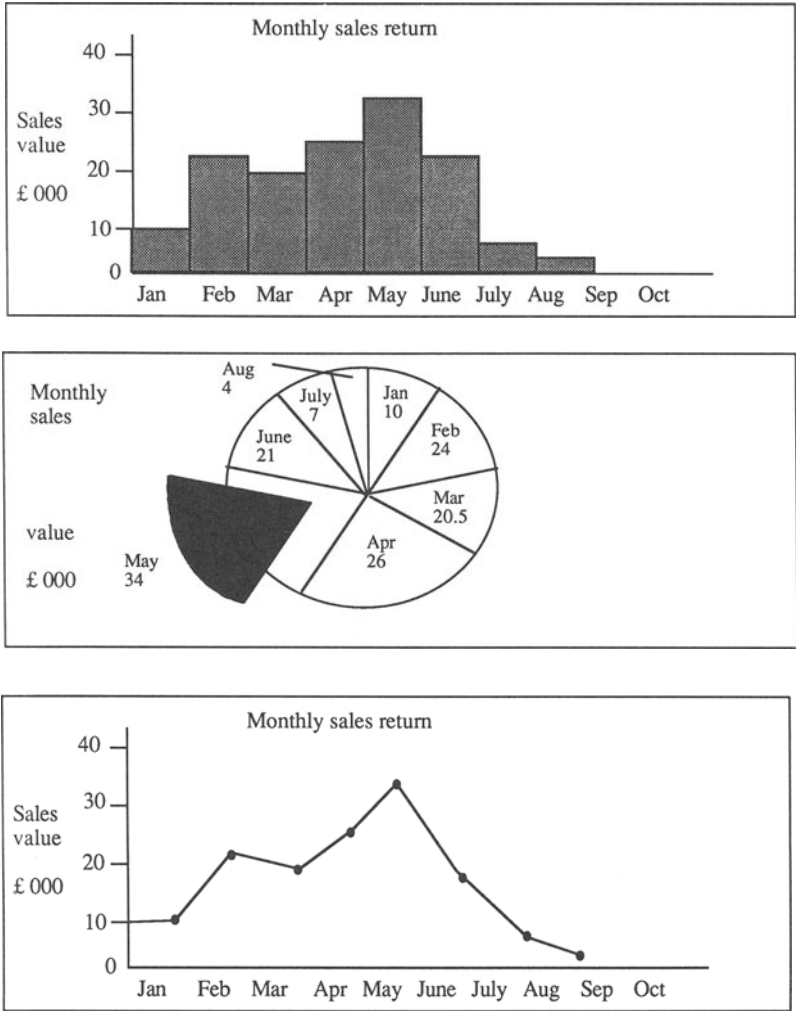
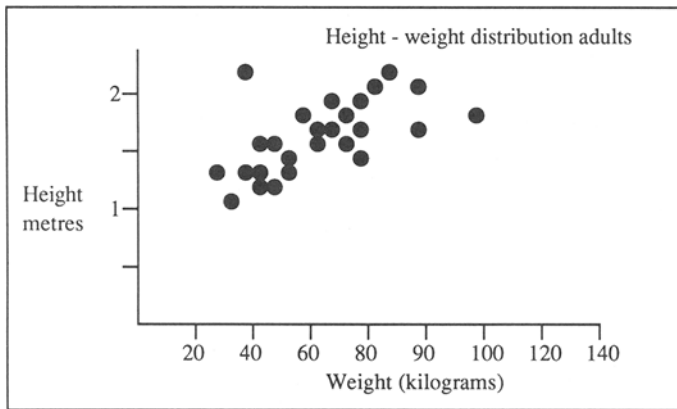


Figure 8.3 Business graphics: example of a histogram, pie chart and line graph.

graphs have the further advantage that they can show more than one measure on a chart, therefore they can be compared; also, associations and differences between populations can be depicted. Generally, the data value is plotted on the vertical x axis, and the range or time dimension on the y axis. Care must be taken with scaling to avoid spikey graphs which hinder comparison and impair the visual impression of trends.

Scatter diagrams. These are used for decimal and integer data on both x and y measures when grouping of items needs to be shown. Items are



(a)



(b)

Figure 8.4 Other types of chart and graphical presentations: (a) scatter diagram of height-weight distribution; (b) three-dimensional techniques—Manhattan diagram.

plotted as points, clustering being apparent from the density of points thereby suggesting a grouping, as illustrated in figure 8.4(a) which plots height and weight measures for a human population.

Three-dimensional displays

When there are three measures per object (x , y and z axes), three-dimensional graphical techniques can be used. Three-dimensional histograms, called Manhattan diagrams (as depicted in figure 8.4(b)), are of limited use even though they can create a striking visual impression. The

problems of scaling and comparison of values shown in perspective means that little meaningful information can be shown unless there is a very marked effect. Multi-variate pie charts can be used by adding more pies to the display to represent the z dimension, but the eye is poor at tracking between pies, making inter-pie comparison poor. Visual short-term memory probably limits attention to one pie at a time.

Three-dimensional line graphs are more effective, especially if grouping measures need to be shown. Supplementary analysis can be added by taking slices through a perspective diagram to show contours on the z axis. Where measures become multi-variate and exceed three values per object, then statistical techniques of factor analysis have to be used to reduce the dimensions to a visual effect which can be plotted, unless the data is expressible in more complex images. These can combine graphics with maps and diagrams.

Other visual representations

Graphs are not the only method of visually displaying information. Hierarchy diagrams are useful for showing categorisation and hierarchical relationships. Sequences, precedence and multi-linked dependencies can be illustrated by network diagrams, and finally use of symbols and icons for simple measures should not be underestimated. Use of icons, chart design and maps can give a distinctive visual impression, as illustrated in figure 8.5 which depicts the capabilities of modern business graphics. Use of symbol size to illustrate a measure can also be effective.

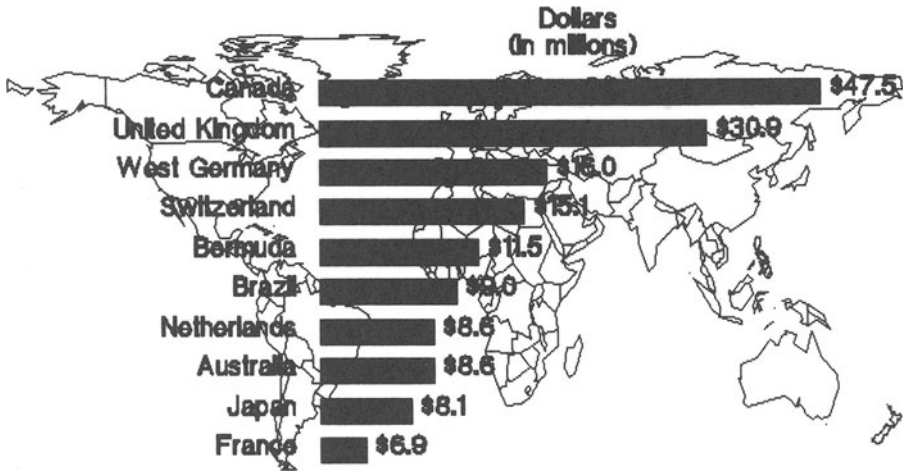
Directions of movement or a trend can be shown by arrows, and association may be represented by proximity in spatial layout or by connecting links. In complex graphs, however, the user will have to expend more effort in interpretation. Wayfinding guidelines should be given to guide the eye through the image; and documentation, supported by help facilities may be necessary to explain complex images.

8.5 Reports

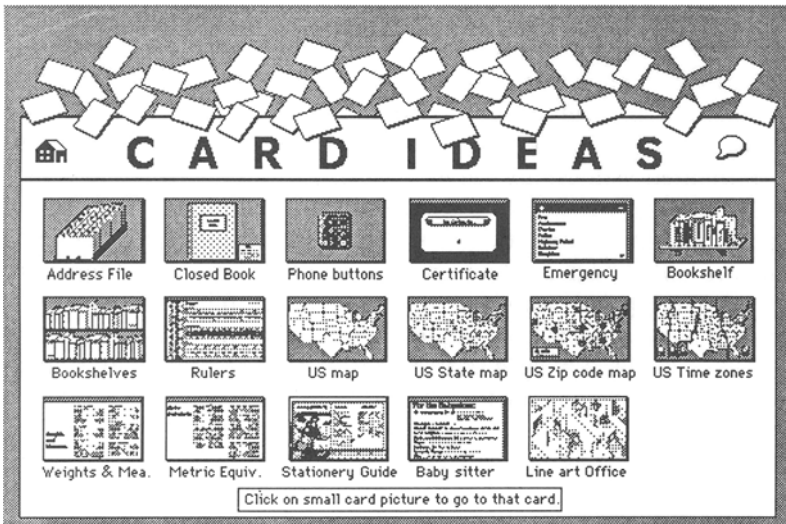
Computer output may in graphical or text form on a variety of media, such as film, paper or VDU displays. This section examines a sub-set of such computer output: printed character-based reports.

During analysis, reports can be classified according to their function and general layout. First the type, function and expected usage of a report should be established. A report's function may be either to convey information from one system to another, for example, an invoice, or to summarise information about a system as in a management summary, or a historical record, simple listing, etc. Whatever the function, most reports

Harrison Oil Company Petroleum Exports



(a)



(b)

Figure 8.5 Use of icons, maps and graphical design in the Apple Hypercards system.

fall into one of three categories: transaction, information and listing reports. The function of a report may influence its layout, however, and general formatting guidelines can be given for all types:

- (1) *Transaction reports*. These contain the results of input data which has been processed and is now being passed out of the system, possibly to be processed further elsewhere. Transaction reports often carry information between systems and contain information about the objects being processed. Examples are order forms, delivery notes, invoices, purchase orders and pay slips.
- (2) *Information reports*. These carry information about the system over the system boundary, and contain data describing system processes and their activity. The information is consumed by managers and system operators to monitor, control and modify the system's behaviour. Exception, monitoring, analysing reports and management summaries all fall into this category.
- (3) *History and archive reports*. These are a special case of information reports when a large quantity of information is needed to describe the state of a system at a point in time (the archive) or information is needed over a long time period to describe a system's history. Processed data which may possibly be required in the future is also held in archive reports.
- (4) *Browsing reports*. These are the simplest report type and are the hardcopy equivalent of the query screen or file listing. Information is generally presented in an unsophisticated form so that users can sift through it in a variety of ways. In the simplest case, these reports are formatted listings of computer files.

Transaction and information reports require most design, although some structure should be given even to archive reports and simple listings. Transaction reports often have similar contents to input forms, and forms design guidelines can be applied. For other reports, general layout guidelines apply as they do to screens and other presentation media. In addition, the following factors should be considered.

Analysis of report usage

The contents required in a report will be specified in output dataflows of the system and in the user's requirements. However the grouping of information into one report can also be affected by other factors which the analyst should be aware of:

- (a) *Frequency of production*: Is the report required on demand or at a specific time, such as daily, weekly or monthly? Is all the information

- required at the same time? Timing requirements may lead to some information being placed in an on-demand report while other items may only be required at weekly intervals.
- (b) *Volume of production*: How many copies of a report are required, and will the same number be produced each run? Reports with different volumes may have to be run separately; hence a long print run will require operations staff to set up printers, whereas a single copy report could be printed without any operator intervention.
 - (c) *Timing and accuracy of information*: When a report is produced can be influenced by how up to date the data has to be and the value to users of information which is not completely up to date. If slightly inaccurate information is permissible (for example, close of business yesterday) then overnight printing of reports is tolerable; on the other hand, information may have to be totally up to date, in which case an on-demand report is necessary. Accuracy also applies to numeric values. Calculation to five decimal places may be needed in an engineering report; in contrast, a cost forecast may be acceptable with an accuracy to the nearest £100.
 - (d) *Security*: This concerns how sensitive the information is and what precautions have to be taken to ensure that it is not seen by unauthorised personnel. This influences the devices on which it is printed and arrangements for distribution.

Layout design

Reports fall into three types of layout designs:

Listings: simple iterations of records, browsing and archive reports use this layout.

Block structure: more order is imposed on the information by rows, columns and totals; information reports are usually block structured.

Group structure: more complex layout with groups of information organised in blocks; transaction reports usually fall into this category.

The steps in report design echo many principles already stated for screen display design:

- First establish the purpose of the report. This should suggest a clear and concise title
- Decide on the report contents in consultation with the user. The contents will be based on the system output specification, bearing in mind the factors listed above under Report usage
- Structure the information into groups and blocks of related items. Grouping may be by data related to an entity in transaction reports, or

by the principle of functional cohesion, that is, data related to one topic or purpose according to the user's criteria

- Order the groups and blocks according to the user's needs and reading sequence, for example, group by importance, cost, frequency of use, sequence of use

Too many reports contain too much data. Overcrowded reports cause longer search times as the reader has to track data items down in a morass of print. High print densities also increase transcription error rates, as figures close together can be mistaken. A report with more than 50 per cent of its area covered by print, including headings and any format characters (such as \$\$\$\$), is overcrowded. Aim for an upper limit of 40 per cent of the total area in print.

Report crowding does pose a dilemma when several pieces of information are required together. If the information is separated on to different pages then the user has to turn pages to find all the data, burdening short-term memory while doing so. Place information on one page and overcrowding may result. There is no ideal answer to this trade-off judgement but, on balance, excessive crowding (50 per cent plus print space occupied) should be avoided.

Listing reports

These are the simplest to design. Data is usually presented in record format as rows organised in columns of information reading down the page. Pages should be numbered and, if the data is ordered in some way, blank rows should be left between sorted groups to help structure the list. Any sorting or order which can be added to the list will help the user to browse through the data. Data fields should be separated into columns and given headings. If all the fields do not fit on to one page width, paginate the report across two consecutive pages to fit in all the fields.

Block-structured reports

Blocks should be ordered using the general design guidelines. Totals should be placed close to and following the data that they relate to. If there is a series of hierarchical totals, a separate summary page should be added showing the progressive aggregation of totals. Variable items are best placed in columns on the right-hand side of the report to prevent a ragged appearance. Many of these design features are illustrated in figure 8.6.

Information blocks should be separated by spaces rather than delimiter characters such as ----- ***** etc., which only increase the overall crowding of the report. Blocks of information should be labelled with a heading on the top left-hand side of the data. This is the location which the

Branch Sales Performance Analysis

Incentive League Table

Branches	Orders/ Salesperson			Order Value		
	Target	Actual	% Perf	Target	Actual	%Perf
Scotland	36.47	35.17	96.44	455530	482939	94.27
North E.	38.01	42.1	110.76	499284	397941	125.47
North W.	38.21	38.43	100.58	545536	530368	102.8
E. Mids	37.06	36.66	98.93	274212	287683	95.32
W. Mids	36.04	35.30	97.92	484105	495872	97.63
West Eng	38.33	35.15	91.72	395938	372924	106.17
W. Lond	32.8	30.4	92.7	436562	485032	90.01
C. Lond	32.7	27.56	84.27	287607	358853	80.15
City	27.66	25.11	90.77	241328	260550	92.62
Anglia	34.81	33.25	95.50	418087	426168	98.1
S.E. Eng	34.12	31.33	91.81	450161	442584	101.71
S. Eng	34.22	31.97	93.44	423471	435733	97.19
U.K.	35.42	33.05	95.58	4980549	5066002	98.27

Figure 8.6 Report layout showing structuring of information by grouping, headings, titles and summaries.

eye first tracks to when reading a continuous text block. Within a system, the report layout should be kept as consistent as possible so that users become accustomed to familiar layouts and learn where to search for information within a report.

Group-structured reports

Transaction reports are the most common group-structured report. The whole report may have a format based on an existing form. If not, the group content and sequencing of data is designed using the general report

design guidelines. Transactions are generally indexed by a unique code which should be clearly marked and placed in the top right-hand corner of the report; this helps leafing through a pile of reports to find the reference number. The report title should be centred and groups of information separated, preferably by space, and ordered according to the criteria cited previously.

Alternatively, separation of information groups may be achieved by using boxes for emphasis or by employing background colour. Excessive use of delimiter characters should be avoided as with block-structured reports. These reports often use pre-printed stationery, in which case it is advisable to print a dummy page first to check on printer alignment of the print fields on the pre-printed template.

Detailed layout

The guidelines are similar to those followed in screen displays:

- (1) The type and format of data items should be examined to determine the number of print character positions required. For example, in COBOL a PIC 9(6) will require 6 positions with one, possibly, for the plus/minus sign. PIC S9V99 will require 5, a mandatory \pm sign, three digits and one decimal point.
- (2) Characters are aligned to the left, numerals to the right, and decimals are aligned on the point. Use of an optional minus sign can give a ragged leading edge on a column of figures. One solution is to make the minus sign trailing; another is to bracket negatives, although this convention must be explained.
- (3) Columns should be separated by at least three blank spaces.
- (4) Headings should be aligned to the centre of columns. Do not adjust column width just to accommodate a heading.
- (5) Highlight important fields, with bold type, different fonts, underlining or colour.
- (6) Number pages and title each one. An unnumbered page which has become detached from a report can be very irritating.
- (7) Date and Time stamp the report at each run. Sooner or later the fact that it was the weekly report before Christmas will be important.
- (8) Finally, when the detailed report layout has been designed, the users' opinions should not be forgotten. As with most aspects of the human-computer interface, users should be consulted about designs and changes made to accommodate their views. In display and report design, early consultation is advisable before the detailed layout is planned, as well as final acceptance testing.

8.6 Summary

Display design involves structuring information to help people read and understand it. Data items should be grouped and groups ordered according to usage. Important data should be highlighted to ensure attention is drawn to it.

Data displays may be either textual or graphic. Text displays should not be overcrowded and space should be used to separate information. Display-control dialogues are designed to help users progress through a body of data according to their needs. In simple cases this will be by scrolling; more advanced access is by data-retrieval/query languages. The syntax and structure of such languages has to be designed with care. Graphical displays make use of human chunking abilities by abstracting the qualities of data. Designers have to choose a chart type according to the data set being analysed and the type of analysis treatment. Common chart types are histograms, pie charts and line graphs. More complex visual representation can be achieved with diagrams, maps and complex images.

Reports should be analysed to determine their function. This may dictate the structure of a report and should suggest its title. Reports have transaction, information, listing or archive functions. Information is ordered and grouped to optimise efficient access. Detailed formatting of columns, text and number has to be designed to make the appearance of a report consistent and pleasing to the eye.

Further Reading

Consult the general references.

9 Computer Control Interfaces

This chapter covers interfaces which control computer operation. These are the familiar menu and command language interfaces of operating systems and any interface which is provided for users to gain access to the system. Following the usual format, general guidelines are given and then control interface types are examined in turn, starting with simple and familiar styles before migrating to more advanced direct manipulation interfaces which are becoming increasingly popular in modern systems.

9.1 Control Dialogue Guidelines

The objective in computer control dialogues is to give users the maximum amount of control concomitant with their abilities, in a manner that keeps the initiative for control with the user and protects the user against making mistakes.

Computer control dialogues fulfil two purposes:

- They give people access to facilities in computer systems
- They allow people to interact with a computer facility, that is, to hold a conversation with the computer to gain information and complete a task

All systems have a first type of control interface; the second is for conversational interfaces in which user and computer interact in simulating behaviour of a system or controlling an external system. Examples of the latter are air traffic control, decision support, chemical plant process control and battle-field simulation systems. This chapter focuses primarily on computer access and conversational control interfaces. A further type of computer control interface allows users to build systems and change the way in which computers respond. These are complex control dialogues of programming languages which also merit separate study in HCI terms and cannot be dealt with within the scope of this book.

To start designing control interfaces we need some basic guidelines. Initiative in control should be given to users; the more initiative provided, the more sophisticated and potentially more difficult a dialogue will be to use. Initiative, therefore, has to be constrained for unskilled users with simple dialogues that present only a few choices. A general rule is that

computer systems should not seize the initiative and force users to perform actions according to the computer's command. In practical terms this means that a dialogue should proceed only when the user wants it to; users should not be locked into options without an escape route, and users should rarely be forced to give replies within pre-set time constraints.

Computer control should be by explicit action on behalf of the user and computer. Implicit actions and 'built in' short cuts in a dialogue may appear to save time but such implicit changes are unlikely to match users' expectations and hence can cause confusion.

Messaging is important in control. Users need to know where they are within the system, so status messages are essential. Users also require feedback from commands they issue to the computer, otherwise doubt sets in about whether the computer has received the command or if it was the correct command to give. The conversation between human and computer should be continuous, like human communication, so that a message from one party is followed by a reply from the other. Gaps in conversation lead to uncertainty and attention being diverted from the task.

Computer control dialogue should be modelled on the user's tasks as far as possible. Although it may be a dialogue which did not exist in the previous manual system, it may have been implicitly present in the way users performed their tasks. They will carry the model of the system in their memory and will expect to see tasks in groupings and sequences with which they are familiar. Failure to model the users' perception of system organisation may result in users hunting for an option in the wrong menu, forgetting a command sequence or getting totally lost within a command interface. Clearly these are scenarios to be avoided.

Commands should always be linked to a single function. Multiple commands for one function only serve to confuse the user and are redundant anyway. As far as possible, commands should be unique and have a good direct link to the function they evoke.

In summary the guidelines are:

- Explicit action by computer and user
- Communicate with the user, give feedback and status messages
- Dialogue at user's pace and initiative
- Dialogue based on user's model if possible
- Single commands for each function

9.2 Simple Control Dialogues

The most simple type of control uses question and answer dialogues in which the computer asks whether a particular option is required or not and the user simply gives a Y/N reply. Slightly more complex examples can move towards a menu-based system. These dialogues, although easy to

use, are tedious after experience has been gained and slow to operate. Because each step has to be answered each time, users can quickly become frustrated with repetitions which they know are just wasted effort. Consequently these dialogues should only be used with naive users or novices who are likely to remain that way.

When using these dialogues, some guidelines to follow are:

- Only one question at a time. Asking multiple questions may seem to be quicker, but the question–answer link will burden the user’s short-term memory
- When linked answers are necessary, redisplay the previous answer. If the previous answer is needed later in a sequence, redisplay it, otherwise errors are caused by short-term memory problems
- Keep sequences compatible with the source document or user model. If there is a precedent for the sequence of questioning, keep to it

9.3 Menu Interfaces

Menus are the ubiquitous computer interface, yet sufficient attention is rarely given to their design. Menus work by users associating a reply code with an option displayed on a screen. Reply codes may be either numeric or characters. Character codes can be mnemonic and suggest the meaning of an option; however, this method has the problem of running out of letters to represent options, for example, the E for edit and E for exit problem. The solution to duplicates is to use a longer code but this hinders the advantage of giving a response in a single keystroke. Numeric codes, although they contain no meaning, are not a hindrance to efficient menu operation.

An alternative to using a reply code from the keyboard is to use a pointing response with a mouse, or to have a revolving band type of menu in which the space bar controls selection by progressively highlighting menu options going down the menu and then back up the top again. The user picks the currently highlighted option with the Return key. The latter method runs the risk of overshoot as users hold the space bar down and miss their options by going too far.

In most systems there are more options than can be easily placed on one menu. This enforces hierarchical organisation of menus. It is important that the organisation conforms to the user’s model of how options and functions within a system should be grouped, otherwise the task of learning the menu hierarchy is made more difficult. Navigation in menu hierarchies presents two problems for users:

- Keeping track within the hierarchy—the ‘where am I?’ question
- Tracing a path through the hierarchy—the ‘where have I been?’ question

To help users navigate, status information about the hierarchical level and part of the sub-system being accessed should be displayed on the top of the menu screen. To improve pathway tracing, a backtrack facility is helpful so that users can page back to the last menu with a single keystroke. A further extension of user control in the hierarchy is to give users 'escape to the top' commands. These design features in a poor menu design, and a better alternative, are illustrated in figures 9.1 and 9.2.

How many options to display on a menu has been the topic of considerable research. There is a trade-off between depth and breadth in a menu hierarchy. Making the hierarchy broad by placing many options on one menu means that users have to spend longer searching through the list of options; however, there are fewer levels of hierarchy to descend. If the hierarchy is made deep with many levels and fewer options per menu then the search time per menu is shorter although the menu level descent time is increased. Intuition suggests there must be an optimal compromise and some studies indicated that this is so, with menus containing 7-9 options being best.

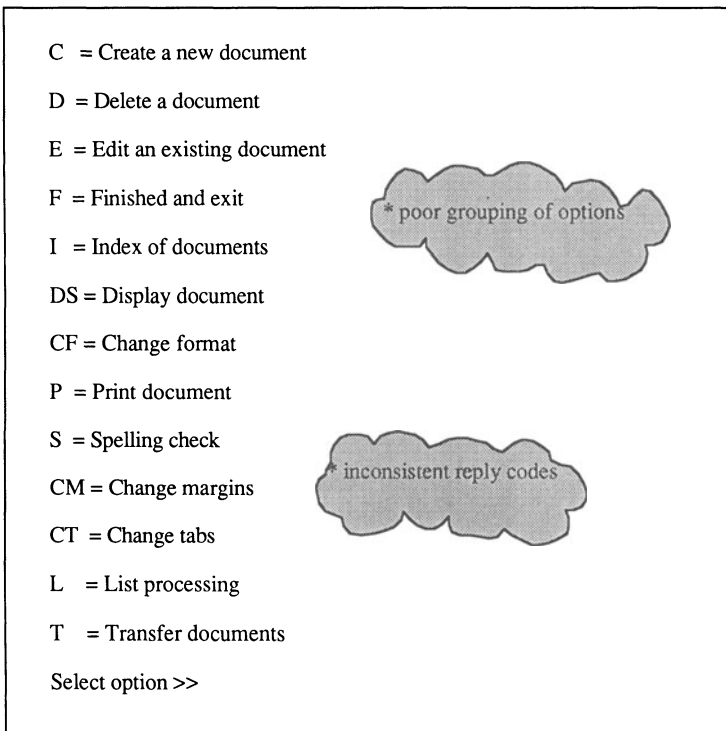


Figure 9.1 Illustration of poor menu screen design. How many more poor design features are there in this menu?

WORD PROCESSING MENU		? for help
No document selected	General options	Office>WP
<u>Editing Commands</u>	<u>Formatting Commands</u>	
C = Create a new document	M = Margins change or set	
E = Edit an existing document	T = Tabs change or set	
X = Exit	F = Font size and type	
<u>File Commands</u>	<u>Sub Menus</u>	
I = Index of documents	L = List processing	
V = View document	T = Document transfer	
P = Print document		
D = Delete document		
R = Rename document		
S = Spelling check		
Select option >>		

Figure 9.2 Illustration of better menu design.

However, the efficiency of broad menus can be increased by structuring the options into groups as exemplified by the WordStar menu (see figure 9.3). In smaller systems the breadth first design may be advantageous because it cuts out traversal time of a menu hierarchy; but for large systems a clear hierarchical structure may be required to help the user comprehend the system, in which case the depth style may be better.

A problem with all hierarchical menu structures is that users soon learn part of the tree and wish to traverse from one option to another without going up and down the hierarchy. To accommodate this desire a menu bypass facility can be designed to give direct access to options. If numeric reply codes have been used, options can be addressed using a page number principle, with the numbers being derived from the menu responses at

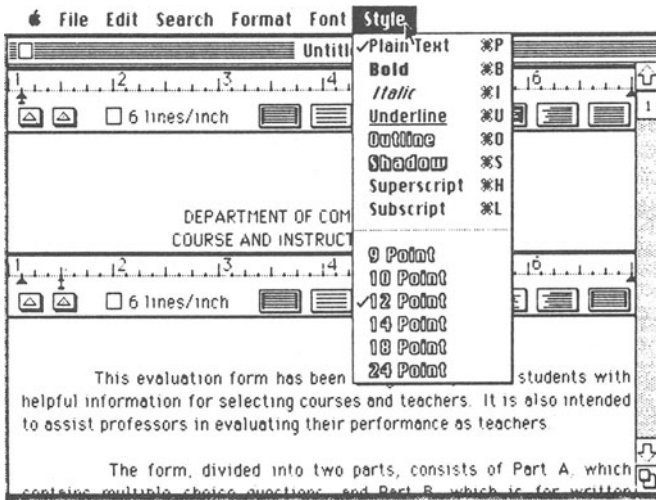


Figure 9.4 Pull-down menu on the Apple Macintosh MacWrite™ program. The menu enables the user to select various character fonts at any step during word processing.

- Bullet-proof the replies, for example if 1 to 7 are options and 0 is escape, make sure any other keystroke is followed by an error message and not a program failure

9.4 Function Keys

Function keys are a hardware equivalent of menus with options allocated to special keys on the keyboard to save screen space and alleviate the reply coding problem. Function keys can either be hard-coded or soft-coded. Hard-coded function keys have an operation permanently allocated to a particular key. The key is clearly labelled with the operation which the user can read; see figure 9.5. This approach is excellent with a single application on dedicated hardware, such as a word processor, when functions are not going to change. For most systems, function keys are soft-coded.

With soft-coded keys the command call is allocated to the function key by the application program. One or more commands can be allocated to each key; but as more commands are linked to a single key, user confusion will mount because of the problem of keeping track of which mode the system is in. In one context F2 key may mean delete a word, in another context it may mean save a file. To help users a partial menu has to be

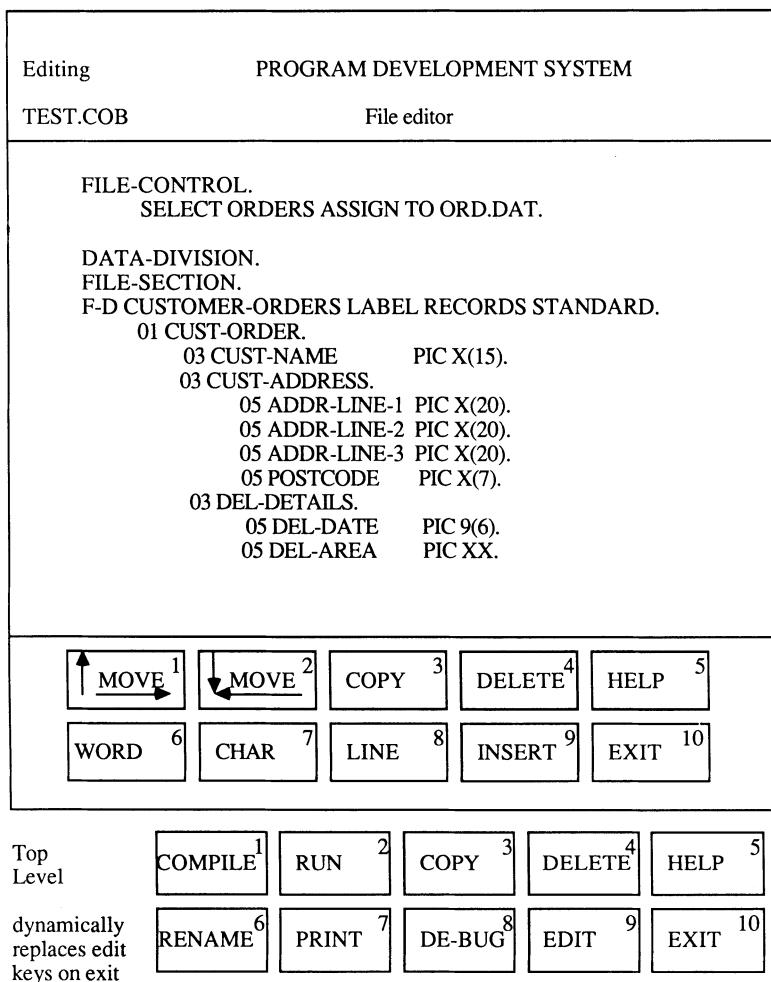


Figure 9.5 Use of function keys for computer control. The screen display mimics the function key positions on the keyboard. Functions may be dynamically allocated by changing the display as the user progresses up and down a menu-like hierarchy.

displayed on the screen showing the allocation of options to keys, mapping the keyboard layout on to the screen.

Even so, function keys can become limited by mode changes. Most computer hardware suppliers provide 10–12 function keys. Important keys should have a constant function in any context (for example, F1 is always

help, F2 is always escape). The remaining keys can be dynamically allocated to 2 or 3 functions each before user confusion mounts. Hence the overall options in a system which can be usefully implemented with function keys are limited.

9.5 Icons

Icons are becoming increasingly popular for representing objects and commands in control interfaces. To be useful, an icon has to be realistic so that a user can recognise the picture and hence the object or command which is being represented. The great advantage of icons is that they are realistic, so we do not have to learn what they represent, and instead can immediately make an informed judgement about their significance. Symbols may also be used; however, symbols initially are meaningless shapes, consequently to be useful they have to be associated with an object. That association has to be learned.

An absolute boundary between symbols and icons is illusory because as soon as a symbol's meaning has been learned it will become a meaningful image. On the other hand, an icon may be ambiguous or have no immediate meaning even though it is a complex and apparently realistic image. Pictorial communication is essentially bound to the interpretation of images made by individual users.

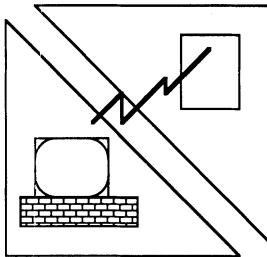
Icons were pioneered by Xerox in the Star system and later by Apple in the Lisa and Macintosh interfaces. A key idea in these designs was that pictures of objects in the system could be used to create a visual impression modelling the user's everyday experience. In this way the Xerox Star system has icons for objects in the office, such as in-trays, filing cabinets, folders, calculators and waste paper baskets. Operation of the system is by picking objects and moving them with the cursor. For example, to delete a file you move a folder into the waste paper bin, following the metaphor of everyday life of throwing waste paper into a bin.

Icons, however, present some problems when functional operations need to be displayed; for instance, cut and paste operations in a word processor, or global find and replace, or check spelling. Some iconic representations can be found such as scissors and a paste brush for the cut and paste metaphor in word processing; but as concepts become more abstract the expressive power of icons wanes. Icons also suffer from problems of ambiguity. One picture may be interpreted in different ways by different people; for instance, the waste paper basket can be misinterpreted as a message basket or as a secure place by novice users. As a precaution against ambiguity, most iconic systems have some text explanation associated with the icons. The problem of ambiguity in icons can be seen in figure 9.6. Poorly designed icons can lead to incorrect

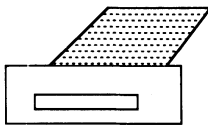
- (a) The Xerox Star in and out tray icons, a clear visual metaphor for messages with text back-up in case of any doubt.



- (b) Ambiguous icon designs



is this a communications link
or an electrical danger ?



a printer or a letter box ?



flight arrivals or a crash ?

Figure 9.6 Design of icons.

interpretation and sometimes to undesirable emotional reactions even though the message is interpreted correctly.

Ambiguity becomes worse when a large number of similar objects have to be represented, or abstract objects and commands have to be illustrated. Try designing an icon for a sort command and the problem should become apparent. As a consequence, the application of icons is not universal, although they have been very successful in creating some interfaces which are very leasy to learn and use for both novice and expert users (learning times of 2 hours were claimed by Apple for the Lisa system).

There is a dearth of guidelines on the design of good icons; however, some advice to follow is:

- Test the representation of the icons with users
- Make icons as realistic as possible
- Give the icon a clear outline to help visual discrimination
- When showing commands give a concrete representation of the object being operated upon
- Avoid symbols unless their meaning is already known

The size of icons is a matter of compromise. If the image is too small then visual discrimination suffers; too large an image, however, consumes valuable screen space. As icons are not a particularly space-efficient means of representation, they run into similar problems as menus with hierarchies. Consequently there is a premium in keeping icons reasonably small. Size is integrally related to complexity of the icon image. Simple icons can be effective in dimensions of 0.5 cm square (for example, the Apple Macintosh window expand/contract symbol); more complex images need to have dimensions in the order of 1 cm.

9.6 Direct Manipulation (DM)

This term was coined by Shneiderman (1983) to refer to interfaces which include icons, pointing and features which have now become associated with WIMP (Windows, Icons, Mouse, Pop-up menu) interfaces such as the Apple Macintosh. The central idea of such interfaces is that the user sees and directly manipulates representations of objects in the system, rather than addressing the objects through an intervening code as in command languages or menus.

Objects are shown as icons which can be addressed by pointing at them with a mouse or another similar cursor control device. Pointing allows objects to be selected. Pointing and selection then invokes a system operation, for example, calls an option as if in a menu or selects a file. Direct manipulation goes further by allowing objects to be moved around the screen using a dragging operation. In this way new associations between objects can be formed, for instance, a file can be placed in a folder (a sub-directory in non-DM interfaces); and operations can be performed on objects, for example, a message is placed in the mail tray. The advantage of direct manipulation is that the computer system models everyday operations more directly than older styles of interfaces; the more directly an interface models reality, the easier it is to learn. This has been proven by the now well established office/desktop metaphor used by Xerox and Apple; see figure 9.7.

The essential features of direct manipulation interfaces can be summarised in a set of principles:

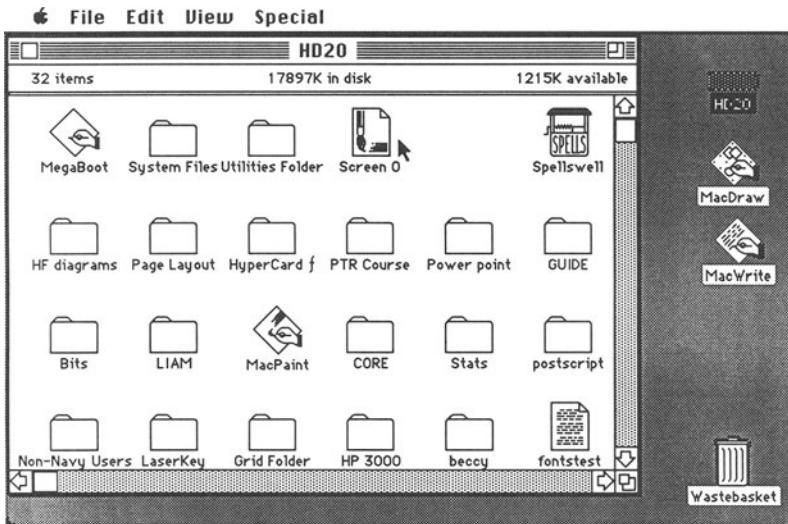


Figure 9.7 Apple Macintosh workstation showing the desktop metaphor.

- *Explicit action*—the user points at and manipulates objects on the screen
- *Immediate feedback*—the results of the user's actions are immediately visible, such as when an icon is selected it is highlighted
- *Incremental effect*—user actions have an analogue/sequential dimension, for example as an icon is dragged across a screen display it moves continuously, following the user's movement of the mouse rather than suddenly jumping to a new position
- *Intuitive interaction*—interaction matches the user's conceptual model of how the system should operate and the display shows pictures of familiar objects
- *Learning by onion peeling*—the complexity of the system is gradually revealed in layers as the user explores system facilities
- *Reversible actions*—all actions can be undone by reversing the sequence of manipulations
- *Pre-validation*—only valid interactions have an effect, so if the user points at an object and this makes no sense in terms of the current task, nothing happens on the display

The interface supports the user's task by portraying a realistic 'virtual world' on the screen. Operation is supposed to be immediately obvious and no error messages are required because invalid interaction has no effect on the interface image. Although such interfaces have undoubtedly been successful and have had a major impact in some products, they do pose

problems for designers and users. In many systems the 'virtual world' has no readily available concrete metaphor to help the designer; also the intuitive model of interaction may be absent if the user is new to the task. The lack of error messages can be frustrating for some users and lead to uncertainty; in addition, learning can be hindered by the lack of explicit representation of all the system facilities. In spite of this, direct manipulation does create usable and, probably of more importance, appealing user interfaces.

The DM idea has given rise to another acronym, WYSIWYG (What You See Is What You Get), which was initially applied to word processors. This refers primarily to the output in which the results of the user's actions are immediately apparent in the display. Old-fashioned text editors have embedded format commands which control the layout of the text, such as '.PP' for a new paragraph. WYSIWYG editors use direct manipulation to format text, delivering the exact image of what the user sees, and hence eliminating the necessity to remember format control commands.

9.7 Windows

Another facet of direct manipulation interfaces is the ability to have several different interfaces at once and more than one view on a single object. Such features are supported by windows. Windows subdivide the screen space so that different operations can be taking place on the screen at the same time. Windows come in two basic types:

- *Tiled*: the screen is divided up in a regular manner into sub-screens with no overlap
- *Overlapping*: windows can be nested on top of each other to give a depth illusion. Complete or partial overlapping is possible and windows can be dynamically created and deleted

Windows have many uses. Screen areas can be separated for error messages, control menus, working area and help. If there are phases in a dialogue when computer control or a sub-dialogue is needed, a control window can be opened and two or more processes can be run at once in different windows. In this manner, windows allow multi-task processing in a suspend-and-resume manner. There is evidence that people work concurrently on several tasks in offices, so windows may be suitable for support of office activities. Windows are also useful for monitoring information. The status of background or suspended tasks can be held in a window so the user can periodically monitor what is going on.

Although windows are very useful they have some disadvantages. If too many windows are created the screen becomes cluttered and mistakes will be made as attention is distracted by something happening in a window not

being worked on. Increased window clutter also incurs the penalty of an unstructured display, and search times increase with complexity.

Use of windows is still a matter of active research, so few definite guidelines can be given for their use. The following tentative advice may prove useful:

- For novice users, simple tiled windows usually suffice; overlapping windows create unnecessary complexity
- Use windows for task swapping (for example, from editing to file management and back again) but keep multi-tasking to a minimum
- Avoid frequent change of the image in windows not being worked on. The changed image will distract the eye and attention from the task in hand
- Delete old windows which are not directly related to the current task. Old windows create clutter

Windows and direct manipulation interfaces require advanced interface software to control the screen display and a high-resolution VDU. Such software acts as interpreter between the application software and the user, managing all the interaction and communication. Interface software of this nature has been termed 'user interface managers' and will be described in more depth in chapter 10.

9.8 Command Languages

Command languages are potentially the most powerful command interface, but more power brings with it the penalty of difficulty of learning. The main advantages of command languages are the economy of screen space, the direct addressing of objects and functions by name (so the need to provide an access hierarchy disappears) and the flexibility of system function which a combination of commands can provide.

All command languages have a word set, called a *lexicon*, and rules which state how words may be combined, which is a *grammar*. The lexical structure of a command is the method of coding meaning into the command words to help recognition and remembering of commands.

Command language lexicons

Command languages need words to identify objects and operations. Objects will be devices, files, etc. which the commands of the language operate on. Objects will usually be described by nouns and operations by verbs. Both word sets should be as meaningful as possible; however, one objective of command languages is brevity of input, hence coding of identifiers is usually necessary. The basic choice when shortening a word is to truncate or abbreviate.

Truncation removes the latter part of a word, leaving a few characters at the front, for example:

DIRectory CATalogue DELete DISplay DEVice

This is an effective technique if the front of a word communicates its meaning. Another advantage is that truncation can be used in two modes. A full version of the word is provided for novices, while experts can use the short form. It is not difficult to write interfaces which can accept both versions. The problem with truncation comes from duplicates between words sharing common leading characters:

DELete DELay DISplay DISconnect

When this happens a further character may have to be added to remove the ambiguity. Unfortunately this violates the consistency rule as users may have to type in either 3 or 4 characters depending on the word. There is a trade-off between risking ambiguity in a command word set and the effort a user has to expend entering commands. Ideally, users should be able to invoke commands with a single economical keystroke; however, single letter commands are more likely to create aliases as the command word set grows. Most operating systems use three-letter commands to prevent ambiguity. Longer commands also improve ease of memorisation.

The alternative to truncation is compression. This uses strategies of code design described in Chapter 6. Characters are removed at various points in the word, leaving sufficient letters to convey the meaning. The resulting compressed words should, as with all codes, be the same length. Simple elimination of vowels or consonants rarely produce good code words; instead, mnemonic techniques of front-middle-back compression using syllabic emphasis give the best results. The main disadvantage of compression techniques is that they cannot be used in a long and short form within one system; also the memorability may not be any better than a truncated code. Consequently command languages have tended to favour truncation coding. Compression codes become more advantageous in large systems with extensive word sets in which truncation is no longer a viable option.

Command language syntax

The rules which govern how command words may be combined vary from simple association rules to very complex grammars. Generally, three gradations in command language complexity can be described:

- *Keyword*: simple command languages which use a single keyword to invoke an operation

- *Keyword and parameter*: a qualifying argument or condition is added to the keyword to make the language more flexible
- *Grammar-based languages*: these are the most complex, in which a full syntactic structure allows very complex commands to be written

These command categories have the following characteristics, although there is no rigid boundary to each category:

- (a) *Keyword*. Command languages which use single nouns and verbs to identify objects and invoke commands. Examples are DIR or CAT to show the directory. Command keywords may be used in very simple combinations, such as the Command/Object construct of a verb/noun pair, for example, TYPE FILENAME, PRINT FILE. No complex grammatic rules are present, consequently the word combinations are limited. The expressive power of the language is dependent on the size of its word set. Early microcomputer operating systems used the keyword method (for example, Apple DOS).
- (b) *Keyword and parameter*. In these languages the basic word may be qualified by added parameters to enhance the behaviour of the basic commands, for example DIR/SIZE, DIR/OWNER, DIR/PROT. This gives more flexibility to the language as one command can now be used to do several different things depending on the parameter. Rules are introduced to govern the set of permissible parameters per command and how they are combined. Unfortunately many command languages add punctuation which is totally redundant and confusing, just to govern the parameter position, for example:

COPY SYS\$STAFFDEVICE:[CNS3013.SYSDIR.LEX]FILENAME.DOC
(The copy command in early versions of DEC's VMS operating system; a bad example of cluttering with delimiters, now reformed)

lpr-Pdiablo myfile (a printer command in UNIX, in which the printer type is the parameter)

Keyword and parameter command languages still have relatively simple rules for combinations of words. The command strings input can be validated by using look-up tables for the command words and valid parameters for each command. The expressive power of the language is greater than a simple keyword language.

- (c) *Grammatical*. A set of rules is introduced to formulate a set of phrases which may be derived by combinations of command words. The rules dictate which word types may occur in sequence within a command word string, just as English grammar constrains the way sentences are

formed. Many command language grammars mimic natural language grammars to help learning, although the types of sentences are simple when compared with natural language.

The types of grammatic constructs required can be grouped in functional categories:

- *Assignment*—this associates two objects, or an object with an attribute or value. For instance, to give a device some property, or to set read-only protection on a file. Command phrases of this type are constructed in the form Verb-Object to Object, for example:

```
SET LINE-PRINTER TO CHANNEL-2
```

```
Assign DiscA =USERfred
```

- *Imperative*—this command invokes system operations and may be qualified by objects for the destination of results using the Verb-Qualifier-Object paradigm, for example:

```
nroff -TLp -ms myfile | Lpr -Plp&
```

(the UNIX command for formatting a file and then printing it in batch mode)

```
stop ws 4
```

(the concurrent CP/M command for stopping WordStar in partition 4)

- *Locate*—commands which search or find a data item within a list or file. Locate commands are common in data-retrieval command languages and take the form Verb(find)Object with Qualifying conditions, for example:

```
FIND CUSTOMER WHERE CSTATUS=NEW
```

(find new customer records)

- *Accept Input*—commands which get input from the user and use that input in an operation, for example:

```
CREATE BACKUP[FILENAME=*****]
```

(create a back-up file with a 6 character name entered by the user)

Analysis of command functions can help selection of more natural syntactic forms for the command. Command languages become much more complex if simple phrases can be built up into more complex expressions. This is effected by a set of rewriting rules which control the nesting of simple

phrases within larger units. In the case of English the grammar dictates the way in which phrases are composed into sentences. The rewriting rules of a grammar can allow many layers of nesting using a recursive principle, so very complex expressions can be constructed. As a result command strings have to be parsed using recursive techniques, familiar to compiler writers, to analyse the segments of a command language sentence.

Command languages with a hierarchical grammatic syntax have the complexity of programming languages and are indistinguishable from them. The other property required for full programmability is the ability to store several command strings together in a file which can then be invoked by its name. Most operating systems provide this facility, allowing users to extend the system's functions by producing new combinations of commands in programs. Examples are COM, EXEC files and shell programs within UNIX. Full syntactic command languages are powerful and flexible, but they impose a considerable learning burden on the user.

Analysis and design of command languages

The functions which a command language has to support should be identified and linked to command names and syntactic structures. Single commands should be provided for each function, as duplicate commands will only confuse users. The level of sophistication of the language should be matched to the users' profile. Generally, full syntactic command languages should be reserved for sophisticated users; however, many users can use complex command languages provided good training and support are provided. If users have a considerable amount to learn, then a layered approach to the language complexity should be adopted. Release a restricted simple set first, then let users progress to the full command set when they feel confident with the simple version.

Command language specification concerns drawing up the command word lexicon and syntax, adding error messages and the help sub-system. Command languages with grammars require specification using the tools employed by compiler designers, that is, BNF formalisation of the grammar or use of syntax diagrams for specification of syntactic sequencing. Error messages should be planned with care. Errors should be anticipated at several levels: lexical mis-spellings, syntactic errors, semantic misunderstanding about usage of a command and run-time errors from the underlying software. The error interpreter should aim to give informative messages which relate to the type of error which has occurred, with an explanation of the most probable source of the error.

Command language design involves design of an input parser, error message interpreter and run-time system. These are systems and compiler design issues which will not be treated further here. In summary, interface design guidelines which should be addressed are:

- Command word codes should be consistent. If EXIT has been used for the escape command, do not use QUIT in another part of the system
- Punctuation and use of delimiters should be minimised
- Entry should be flexible and forgiving. Double spaces between words should be ignored and mis-spellings corrected if possible
- Command language words and syntax should be economical. Use of the smallest combination of words for a function should be traded-off with word clarity for ease of learning and remembering
- Command words and syntactic sequences should be natural and familiar; for example, use COPY from fileA to fileB, and not PIP destinationfile=sourcefile
- Limit unnecessary complexity. The larger the lexicon and the greater the number of grammatic rules, the harder the language will be to learn. Eliminate duplicate rules and synonyms
- Allow editing of the command string rather than requiring the user to retype it

9.9 Natural Language

Natural language has been heralded as the ultimate type of human-computer dialogue. Although language is undeniably the most natural way to communicate with a machine, in practice natural language appears to create some problems in interaction. Furthermore, machine understanding of natural language is one of the most significant challenges of computer science research, and consequently practical natural language interfaces are still in their infancy.

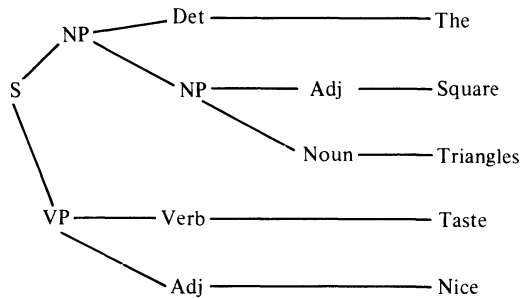
Natural language, like command languages, consists of a lexicon and a grammar. Unlike command and programming languages, natural language has many more rules for syntactic composition which allows more flexible expression and ambiguous interpretation. This section does not aim to cover the complexity of natural language understanding in depth; instead, the aim is to give an impression of the complexity of the problem and then guidelines for use of natural language interfaces in view of those problems.

Syntax

Language is composed of words (called 'lexemes' in linguistic jargon), which can be classified into nouns, verbs, adjectives, etc. Grammatical rules state how word classes can be combined to make well formed sentences, for example, 'He must go to the station to catch the train' is correct English, whereas the equivalent in German is 'He must to the station the train to catch go'—'Er muss nach dem Bahnhof den Zug erreichen gehen'. The rules state the composition of sentences in terms of sub-components,

noun and verb phrases, which in turn are composed of word classes. The composition rules vary between languages and can be very complex. English has approximately 20 000 rules and grammarians estimate that the known rule set is not complete.

Sentences can be decomposed using parsing strategies which test a sequence of words against the permissible combinations. Parse trees are a commonly used representation of the syntactic structure of sentences. Syntax, however, can only tell the listener whether a sentence conforms to the grammatical rules of a language. To generate meaning from language another dimension is needed, called *semantics*. To illustrate the point, syntactically correct phrases can be constructed which are obvious nonsense; for instance, 'the square triangles taste nice' is clearly meaningless yet the sentence parses correctly:



Semantics

Semantics is concerned with generating meaning from knowledge about words and the associations of words. It forms the link between language, memory and experience. Semantic rules can be built into grammars in an attempt to eliminate nonsense sentences; however, not all semantic rules and classifications are exact and building a complete semantic grammar is very difficult. Many words are lexically identical yet have different meanings, such as Bank (as in 'put money into') and Bank (as in 'embankment by the river'). Semantic ambiguity forms the basis of puns in which two meanings can be applied to the same sentence, as in the following example from an encyclopaedia entry for Nell Gwynne, one of King Charles II's mistresses:

Gwynne, Nell—see under Charles II

Semantic ambiguity can be compounded further by inadequacies of syntactic rules. An example of syntactic incompleteness is the lack of any

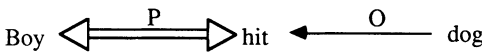
scope rules in natural language grammars. Programming languages are full of such rules which define the structure of programs in terms of control constructs, for example, While . . . End-While, If . . . End-If. Natural language has few scope rules. Consider the sentence:

Jane’s mother put the birthday cake on the special jubilee plate because she knew Jane would like it.

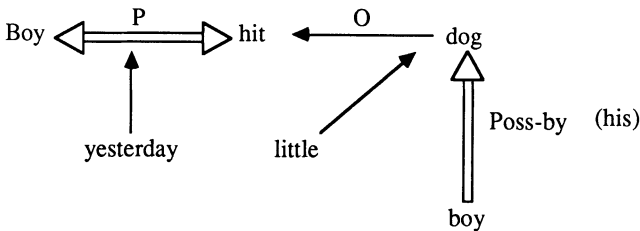
Did Jane like the cake or the plate? The sentence is ambiguous because the pronoun reference has no scope rules about how far back it can refer.

Semantic meanings can be analysed using network diagrams (see figure 9.8) which constrain the types of words which should naturally be placed together. Semantic networks are rarely complete so it is impossible to specify exactly the meaning in the variety of sentences possible in natural

(a) The boy hit the dog



(b) The boy hit his little dog yesterday



Verbs and nouns are related according to a set of semantic primitive forms denoted on the connecting arcs, e.g. P = Ptrans, a transitive verb with physical contact, O = object of verb, Poss-by shows a possessive clause (after Schank and Abelson, 1977)

Figure 9.8 Conceptual dependency network showing relationships between components of a sentence in terms of their meaning.

language. However, to understand language people do not rely on semantics and syntax alone. To generate more complex understanding, the context of the speaker and listener is required. This is called the *pragmatics* of language understanding.

Pragmatics

Consider the statement 'He is supporting the reds'. This could have a very different meaning depending on whether the reader was aware that the speaker was referring to a communist party meeting or a support of Liverpool football club. Pragmatics is the application of knowledge about the speaker, the surroundings in which communication took place, with other factors such as gestures made by the speaker and past experiences of interactions between speaker and listener.

From the above discourse it should be apparent that language understanding is a very complex matter which is inextricably linked to knowledge about the world and the meaning of words. While computer systems can be built to parse sentences successfully, constructing a true understanding machine is more difficult. One approach is to equip the computer system with a lexicon of words and associated facts, so the parser can resolve ambiguity and avoid errors of misinterpretation. The power of such systems is dependent on the size and complexity of their lexicon, which rapidly becomes a vast knowledge base, unless the domain of knowledge can be restricted. Therefore, most practical natural language systems restrict understanding to a small specialised area of knowledge, so the lexical/semantic knowledge base can be constructed with a fair expectation that it will be reasonably complete.

Problems with natural language interfaces

People are over-ambitious in their assumptions about machine intelligence. They tend to expect computer systems to understand complex sentences, incomplete and ambiguous utterances as they would use with their fellows. This projection of human qualities of understanding on to computers means that users quickly reach the limits of the system's abilities and misunderstandings occur.

Apart from the inadequacies of language understanding systems, there are further problems caused by the inaccurate way in which we use language. Consider the statement attributed to a nameless judge: 'It takes no training to distinguish between the false and that which is untrue'. Many people would automatically correct the error and restore the sense of the statement by altering untrue to true, probably unconsciously. Computer programs have great trouble in making such inferences. Other inaccuracies and ambiguities are frequently found in the following constructs:

- *Statements of time*—the ordering of events is often unclear, for example, read the following instructions:

Please sign the list for a taxi.
 Registration forms should be handed in at the desk.
 Please fill up the seats in the centre of each row first.
 Before entering the dining room please wash your hands.

It is not clear how long the time interval should be between the actions and which instructions should be executed first if the sentences are in a list. Knowledge that one has to register before entering the conference may help. Likewise statements such as please telephone immediately, as soon as possible, or soon, all have different meanings for individual people

- *Quantifiers*—words such as many, some, often, sometimes, are all vague. Their meaning is derived from the reader's knowledge about the subject being discussed. Two readers may ascribe very different values to the same quantifier in the same context; for example, 'Some students are lazy'—is it 0.1, 1, 15 per cent or more in your opinion?
- *Logical operators*—English and most other languages do not distinguish between the computer inclusive/exclusive OR, consequently people use 'or' when they mean 'and' and vice versa. Consider the statement:

A large vehicle is one considered to be over 32 feet long or 9 feet 6 inches wide or 38 tons laden weight and not licensed to carry passengers.

Does the large vehicle possess all three initial properties, or only one, and does the not condition have to be true?

- *Numeric comparisons*—logical operators of the type 'greater than' are confused with 'greater than or equal to', for example:

Find all staff with salaries more than £20 000. Most people would include staff with salaries of exactly £20 000 as well

Design of natural language interfaces

In view of the substantial problems of natural language processing, the main guideline is restrict interaction to a small domain of knowledge. With current technology, successful natural language interfaces can be constructed using single word recognition, as this does not have the problems of syntax and largely ignores semantics. Understanding sentences is more difficult but possible in restricted circumstances. A successful natural language database interface product, called Intellect, builds knowledge of

the database entities, attributes and synonyms into the interface, enabling natural conversations to be held.

With the current state of the art of language understanding systems, it is advisable to:

- 'Back translate' the user's input so that meaning can be clarified by a further dialogue. This is helpful in reducing ambiguity in statements
- Design dialogues to obtain values for linguistic quantifiers such as 'some, many, more'
- Interrogate users about new terms so that they can be incorporated into the knowledge base
- Avoid giving the impression of understanding too much, or outputting statements which imply reasoning

Speech

If natural language is input by a keyboard it loses much of its advantage as an interface technique because it is too verbose. Users spend too long typing in a sentence and make mistakes. Speech understanding is therefore a natural extension of language understanding which harnesses the full power of natural language interfaces.

Unfortunately, speech introduces the further problem of deciphering words from a nearly continuous physical sound (see chapter 2, section 2.3). People are often ungrammatical in written communication; and in speech they are even more lax with the use of language. Spoken communication is full of unfinished phrases, ungrammatical sentences and mispronounced words. Furthermore, words are spoken in a variety of dialects and speakers use intonation in the voice to convey meaning, for instance:

Find the glass—this can be a question with a meaning 'have you found the glass?', or it can be an order with a different intonation.

Speech recognition systems have to deal with all of these problems. Simple systems have been available for single word recognition for a number of years. Initially systems could only recognise non-dialect speakers but more recent systems can learn the tonal qualities of a speaker. Continuous speech is more problematic. The translation of some words from phoneme (sound) to lexeme (written word) is unfortunately dependent on understanding meaning, either because one word sounds the same as another, as in boar and bore, or because of mispronunciation, or because one word is used with different meanings, such as Bank. Speech is transient and if the meaning cannot be deciphered quickly the system rapidly becomes overwhelmed with more input. With current technology, real-time language understanding with speech is not possible. However, experimental systems

can demonstrate continuous speech understanding of simple sentences, and complex ones in limited areas of knowledge.

9.10 Summary

Command interfaces are designed to give people access to system facilities and to control the computer's operations. Basic guidelines are concerned with giving the user the correct control and helping navigation in systems.

Simple command interfaces use question and answer dialogues or menus. Menus need a hierarchy in large systems which makes access slow. Bypass techniques can be helpful for experts. Breadth-depth trade-offs can be made and menu formats designed to optimise recognition. Function keys are a hardware-assisted menu design which economise on screen space but are limited by the number of keys provided.

Icons, windows and pop-up menus are all part of direct manipulation interfaces which work by users picking and interacting directly with objects via a screen image metaphor instead of an identifying code. The screen presents a virtual world based on the users' view which contains an intuitive metaphor to guide interaction. Icons have limitations of realism and ambiguity in large systems. Windows provide multiple views on tasks but may be distracting if overused.

Complex command interfaces use command languages or natural language. Both consist of words and composition rules for sentences, called a grammar. Command languages have relatively few grammar rules but can provide a powerful and flexible interface. Care has to be exercised in choosing abbreviations for command words. Natural language has a vast number of composition rules, but in spite of this, it is still inherently ambiguous. People decipher meaning using semantic and pragmatic knowledge. The inability of machine systems to store sufficient knowledge for sophisticated understanding means that users expect too much of language interfaces and tend to exceed the system's capabilities. Limited natural language interfaces can be used for dialogues about restricted areas of knowledge.

Further Reading

Shneiderman (1987) gives a good survey of command and control interfaces and deals with direct manipulation in more detail.

10 Development of Human–Computer Interfaces

This chapter covers a series of topics which relate to research issues in interface design. First interface design is reviewed within the wider context of system design and prototyping, which is followed by examination of the evaluation of human–computer interfaces. Some approaches to the problem of interface design for different types of users are reviewed, that is, the concept of adaptive and intelligent interfaces. Within this topic, user interface software (generally termed ‘user interface managers’) is reviewed. Finally, formal specification methods for interface design are described, concluding with a discussion on future developments in human–computer interface design.

10.1 User-centred Design

It has already been emphasised that interface design is part of the system design process and should be integrated with current system development methods. Unfortunately system development methods have paid little or no attention to the problems of interface design and, so their critics would maintain, to the users themselves. A common theme within interface design is concern and involvement with users. A group of methods have been developed, partly within the human–computer interaction community and partly within the area of systems science, which aim to improve the human involvement in systems development. These methods advocate the following approaches to the design process:

- (a) *User-participative design*: Users should be actively engaged with the process of design and should be assigned to the design team to share in decision making. This is intended to narrow the gap between computer specialists and computer users and to help eliminate errors in communication which result in poor requirements definition. Critics point out that in practice ‘user experts’ get themselves elected on to the design team and become part of the system design community, thus perpetuating the user–specialist division.

- (b) *User-centred design*: The system design should be driven by the needs of the users and not by functional processing requirements, limits of hardware, etc. All good methods of systems analysis should focus on the user's requirements. Emphasis on task analysis and design helps user-centred design; however, beyond exhortation for good practice, there is no prescriptive method one can apply to ensure user-centred design.
- (c) *Iterative design*: The concept of prototyping and cycles of refinement during design is frequently urged in the human–computer interaction literature. Early design stages are described as formative when the broad design features are specified and prototyped; the product then goes through stages of summative design in which details are added and improved upon. While prototyping works well when interface operation is not complicated and prototyping tools are available, it is more difficult for complex interface designs which stretch or exceed the resources of prototyping tools. It is in just such systems that the interface is likely to be critical.

Prototyping is undoubtedly helpful but there are practical limitations to its applicability. In many cases it is essential to build a complete system to create the necessary interaction before judging a prototype. Also prototyping can lead to poor specification in which problems are deliberately avoided with the excuse that the answers will be found during prototype trials.

Each approach undoubtedly has something to offer in improving systems and interface design. User support can encourage better user-centred design by involving users with system operation as well as in the design process. Local experts can be recruited from the user community to act as semi-expert advisors on the system after training has been completed and the implementation team has departed. Local experts can increase commitment to the system as well as providing a human help system, although developers should beware of demanding too much from a single local expert.

10.2 Evaluation of Human–Computer Interfaces

Evaluation of human–computer interfaces should be carried out in conjunction with prototyping development and on complete products produced within the more traditional analysis–design implementation life cycle. Interface evaluations vary considerably in the approach taken, the method of data recording and the treatment of results. Broadly three approaches may be followed:

- (a) *Diagnostic analysis*: This aims to pin-point the poor design features in an interface design in an intuitive manner by examining recordings of dialogue sessions. These are usually videoed and then inspected for signs of user frustration, users' errors and misconceptions.
- (b) *Monitoring*: Interfaces may be evaluated by monitoring one or more features of their usage such as error rates, frequency of command use and duration of usage. Monitoring may be carried out by logging system commands and terminal input–output signals with operating system facilities or by specialised line monitors.
- (c) *Experimental analysis*: Experiments are designed to test empirically two different interface designs or two different features of a design. Experiments control the context of interface operation carefully to give precise results about the design under test. Data may be recorded by a variety of techniques.

Measures of evaluation may be either objective, that is, derived from controlled collection of data, or subjective, that is, based on intuitive judgements and opinions gathered from users. Diagnostic analyses, while pin-pointing critical features of an interface design, are a subjective approach. Monitoring is more objective although lack of knowledge about the context in which the measures were collected may lead to difficulties in objective interpretation. Experiments which control the context and use empirical measures are the most objective measure but pay a penalty in the small number of features which can be measured in any one experiment. Objectivity, however, does not just depend on the approach but also on the method of data recording. The following techniques may be used to gather evaluation data:

- (1) *System logging*: Recording the input–output traffic between user and computer. This is non-intrusive in the sense that the user is not disturbed by the measuring and it yields data for objective analysis.
- (2) *Video recording*: An interactive session is videoed and subsequently analysed by playback for intuitive diagnosis or by collecting more objective behavioural data from the recorded film. Video recording may be unobtrusive if the camera is hidden behind a one-way screen.
- (3) *Direct observation*: The observer sits beside the users and records what they do by a tape-recorded commentary or writing interaction details on check sheets. This type of recording intrudes on the users because it is difficult to ignore the observer's presence and this can lead to bias if the user is distracted.
- (4) *Protocol analysis*: Users are asked to think aloud about what they are doing in terms of mental activity, decisions and reasons for decisions. While this method is intrusive, it is one of the few ways of getting a record of the user's mental activity during interface operation. Data is

- open to the user's subjective interpretation about what was happening.
- (5) *Questionnaires*: Questionnaires are useful for collecting subjective data and some semi-objective data about user characteristics. This technique is necessary to discover users' attitudes and opinions. Questionnaires can be augmented by interviews to gain further understanding about particular points such as misunderstandings about a dialogue and difficulties in using an interface.

Analysis of data can either be intuitive or quantitative. If quantitative measures have been collected, a variety of statistical techniques exists to help the evaluator assess the results. For further details the reader is referred to Robson (1973).

The above approaches to evaluation have assumed that a product, or at least, a prototype exists. However, sometimes it may be advantageous to evaluate a design before it is built. Predictive evaluations of this kind specify an interface dialogue in terms of a grammar and then analyse the grammar phrases for the number of words (that is, commands) and grammatic rules (combinations of commands). The more words and rules a dialogue has, the more complex it is and the more difficult it may be to learn. The best developed of these techniques is the action grammar of Reisner (1984).

The selection of recording techniques, approach and analysis techniques depends on what the evaluator wants to measure. There is no ideal measure of a good interface but some ideas are beginning to emerge about what are the important qualities of an interface from the users' point of view. These qualities have been christened with terms such as 'acceptability' and 'usability' and have the following components (after Shackel, 1986):

- (a) *Utility*. This is a measure of how well an interface (and the system) helps the user to perform one or more tasks. It is linked to the functionality of the system (what you can do with it) and the task fit, for example, how well does the interface facility provided match what the users want to do and their perception of how to do it (the task).

This is difficult to measure. Attitude data from questionnaires can give some feel for task fit, but more comprehensive analysis requires elucidation of a user task model.

- (b) *Effectiveness*. This is a measure of how well the interface, and hence system, performs in achieving what the user wants to do. This can be measured in terms of:

Error rates lower than a target level.

Task completion time within a set target time.

Usage of system facilities above a minimum target frequency.

- (c) *Learnability*. This measures how easy to learn a system is, and how well it is remembered after a period of disuse. Learnability can be quantified with measures of:

Decreased error rates over time from the start of system usage.
Decrease in task completion time from the start of system usage.
Correct recall of system facilities, operational procedures or command names.
Increase in user knowledge about system facilities over time.

- (d) *Coverage*. Coverage is the quantity of system facilities that are used. While not all users can be expected to use all parts of the system all the time, if some facilities are never used by any users there may be design problems. Coverage is measured as:

Facility usage by x per cent of users within a set time period.

- (e) *Attitude*. Attitude is the subjective part of usability which quantifies user satisfaction with the system.

User satisfaction exceeds a target rating.
User-perceived problems are kept below a set level.
User motivation to use the system exceeds a set baseline level.

All these measures require goals to be set based on reasonable expectations for the system before the evaluation is carried out. Most evaluations have concentrated on a small number of measures, typically task completion time and error rates. While these measures can give an overall impression of an interface's usability, finding out why an interface has usability problems is often more complicated. For instance, it may be found that help screens in a system are rarely used. There are five possible interpretations for this observation:

Users found the help screens so good that they only needed to use them once or twice.

Users found the system so easy to use that they rarely needed to refer to the help screens.

Users found the help screens so bad that they gave up using them after an initial attempt.

Most users did not know that help screens were in the system.

Users found the command to access help screens difficult to use.

To find out the answer to this question, data collected from logs would have to be combined with questionnaire data, and even an experiment may be necessary to quantify how useful the help information was.

Evaluations, in conclusion, can be done simply to get an overall impression of how good an interface is, and recordings can be inspected intuitively to diagnose problems; but teasing apart the reasons why an interface design is poor is more difficult.

10.3 Adaptive and Intelligent Interfaces

One of the central dilemmas of interface design is how to satisfy the conflicting demands of different users, in particular, novices and experts. Novices require easy-to-use, supportive dialogues; experts on the other hand need quick, efficient dialogues with less support. However, with practice most novices become experts. The choice is to adapt or not to adapt.

Adaptability in interfaces unfortunately implies change to some part of an interface design. Change offends the consistency principle and makes the user less sure of the interface, to say nothing of having to relearn parts of it as it changes. The quest to solve this problem led to the notion of adaptive interfaces. The problem is threefold:

- Measuring the user in order to determine when to change; the interface must monitor the user so that it can determine that the novice is now an expert, etc.
- Adapting the dialogue so that it responds to changes in the user's needs
- Making sure the quantity and type of change does not cause too much inconsistency in the interface design.

One simple approach to the first problem is to let users decide about their needs. Users are good judges of their skills; therefore, if an interface has a level switch built-in to change the sophistication of the interface design, then users can elect to switch the interface into expert mode if they so wish. Unfortunately switches of style tend to create considerable inconsistency because a new dialogue style is suddenly presented to the user. The new style has to be learned which discourages people from changing levels.

If adaptation is not user-driven then the problem is how to measure the user's abilities. This presents the same problems as any evaluation (see section 10.2) compounded by the limitation that the interface can only collect data by system logging. The intelligent interface has to try and figure out how sophisticated a user is, based on simple measures such as error counts, command usage and task completion time. Task operation, however, can be affected by mistakes at the lexical, syntactic or semantic level. A user may make a mistake in a command string either because of a simple syntactic error of mis-spelling a reserved word, or a syntactic error in word order, or a semantic error of entering a correct command for the wrong task or in the wrong context. Deciphering these possibilities requires subtle evaluation.

To make decisions about the user, the system has to have a model of the user. This model may be a general model in terms of user skill, driven from error rates and task completion time. Also included may be a knowledge model of how much a user knows about a system which is determined by command usage statistics. As user exploration of the system increases, so the model assumes more knowledge, and this triggers the adaptive interface into providing the user with more facilities. The problem lies in trying to find the correct level of triggering and then the link between monitored data and inferences about the user. For instance, a user may use an advanced command once or twice out of curiosity but subsequently never use it again. A frequency monitor may pick up the user's experimentation and decide that the user therefore has expertise.

Even when the interface has deduced how skilled its user is, the problem has not been solved. The next question is what part of the interface to change. Change of the dialogue style can present problems of consistency; although if different styles of task operation can be detected then it may be possible to match user type to the task style; for example, experts often take short cuts to complete a task whereas novices will go through each step. This approach implies the system has a task model to match different levels of expertise. A safer part of the interface to change is the support components. Messages, prompts and help screens can be very detailed, providing long explanations for novices or concise messages for expert users. Skilled users often ignore over-verbose messages in dialogues; consequently adaptive interfaces should be able to match the messaging to the users' abilities. This adaptation does not change the dialogue style so there is little inconsistency in the change.

Adaptation remains an issue of contention and is the subject of considerable research activity. How much adaptation is a good thing and how well adaptation can be linked to the user's abilities are problems still to be solved.

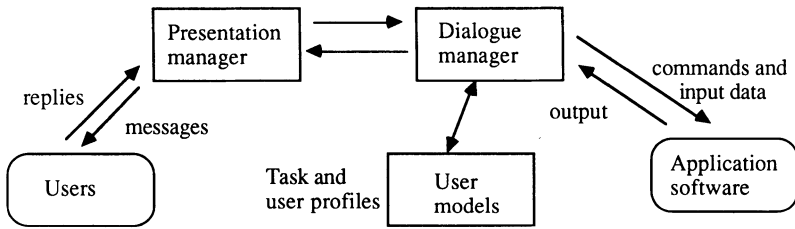
10.4 User Interface Managers

Adaptive and intelligent interfaces are one type of user interface manager (UIM). Such systems are a self-contained piece of software which takes over all the functions of managing the user interface, leaving the applications software to get on with the job of processing. The motivation for UIMs is simple. Given that most software is now written for interactive systems and that a large proportion of code is written to implement the user interface (up to 80 per cent in some estimates), it follows that the same thing is being rewritten thousands of times, usually with no improvement. A UIM intends to be a flexible, re-usable interface module which

communicates with the applications software on one side and with the user on the other side, as depicted in figure 10.1.

The UIM is responsible for all interface presentation and dialogue management, such as displaying screens, accepting and validating input, issuing error messages, providing help and tutoring systems. When the applications software requires data from the users for either a decision or in the form of transaction data, it sends a request to the UIM which then communicates with the user to obtain the data. When the data transfer has been completed the UIM returns the data to the applications software. Although simple requests can be passed between UIM and applications

(a) Modular view, in which the presentation manager handles low level interaction at the interface, while the dialogue manager takes strategic decisions about dialogue content and conversational control.



(b) UIM link module concept. All interface tasks are handled by the UIM and parameters are passed via link modules which have no knowledge of the underlying software.

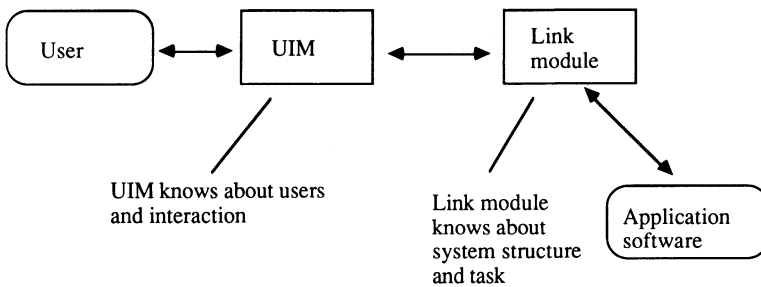


Figure 10.1 Schematic diagrams of User Interface Management systems (UIMs).

software by parameters, many operations require the UIM to have knowledge of the applications software and the task. In validation, for instance, the UIM must know what range of data is valid from the application software point of view, and in a complex task (such as air traffic control) the UIM has to keep track of the interaction. This creates problems in trying to separate the application from the interface.

Separation can be increased by a linkage module which knows about the application on one side and the user on the other. Change to the dialogue can thus be isolated in the UIM and linkage modules, leaving the application unchanged. There are levels of complexity which UIMs have to deal with in human-computer interactions; at the lexical level of key-strokes, separation is simple. Validation of lexically correct replies can be effected by look-up tables. The next level of complexity is the dialogue syntax; separability can still be maintained because input and error-handling sequences are low-level features common to nearly all applications. At the task or semantic level, separability becomes more problematic because the UIM has to have an embedded user task model to be able to react appropriately to the user requests. It is debatable whether the model belongs in the UIM or the applications software.

The power of UIMs depends on their flexibility to accommodate different types of applications software and their portability between different system environments without substantial modification. To fulfil the objectives the UIM has to be as independent as possible from the applications software. The problem becomes a little harder when direct manipulation is used. The interface knows only about mouse movements and icons on a screen. To make sense of pointing responses, the software—either the UIM or the application software—has to interpret the pointing coordinates against a physical screen layout and a logical screen definition file to decide what action to take. If a UIM is to be responsible for interface management it has to know all the operational pathways through the applications software which the interface may have to represent.

This necessitates that the UIM has to have a specification of the application software behaviour. Furthermore, considering one objective of UIMs is to improve interface design, it has to know about good methods of interacting with the user. The UIM needs a constrained set of interaction possibilities based on good human factors principles, so bad interface designs cannot be implemented. Both of these concerns lead towards the need to specify formally and perhaps standardise interaction, a topic covered in section 10.5

The all-embracing UIM is still a research topic and much debate centres around how separable application software is from the interface (see Cockton, 1987). An alternative to the complete UIM is to provide interface building and managing tools which help the applications programmer to construct interfaces quickly and evaluate the designs within a

prototype development cycle. Simple tools, such as screen painters, allow quick development of menus, form-filling screens, data display screens (for example, DEC's FMS—forms management system—or dBASE III screen generator). Most 4th generation languages include screen design facilities.

Advanced workstations require user interface management software to interpret pointing responses and handle window displays. Such interface systems generally allow definition of a logical object file for screen images, drawing physical images such as icons and symbols, combined with window and pop-up menu management. However, it is still possible to design bad interfaces using tools. More active tools are required which encourage good design practice and help the system designer to achieve usable and efficient interfaces. Some systems are available that go some way towards that end, for example, ZOG (Aksyn and McCracken, 1983) and USE (Wasserman, 1984).

10.5 Formal Specification of Dialogues

Software designers have been interested in formal specification to help improve the reliability of designs by being able to prove facts about software behaviour. In human-computer interaction the quest has been to formalise dialogue behaviour in similar terms. If the interaction can be formally described, two benefits can arise. Firstly, the software which implements the interface can be designed with similar formalism and thereby, it is hoped that it should be more reliable. Secondly, if basic principles of human-computer interaction can be formalised, then software modules can be built to implement them. This would be like freezing good dialogue guidelines in re-usable software modules which can then be incorporated into any number of interface designs.

Several methods of formalism have been adopted. Languages already used by computer scientists, such as Z, may be used to formalise interface behaviour (Sufirin, 1986). Other approaches have used path algebras (Alty, 1984) to describe dialogue sequences and algebraic formalisms to describe a set of interactions for a system (Dix and Runciman, 1985). The techniques of formalism differ in their expressive power and flexibility; but all aim to demonstrate a finite description of dialogue behaviour. Besides the debate about the power of the formalism, the critical point is what to formalise. Many of the guidelines derived from psychology are heuristic and dependent on context for interpretation. This makes formalisation in their current state of precision practically impossible.

One answer to this has been to look for generalised principles of interaction which may be reliable enough and context free to allow formalisation. Rules could then be generated for each context of a design rather than a large rule set which tries to account for all the different

contexts of interface designs. Generative User Engineering Principles (GUEPS, Harrison and Thimbleby, 1985) have attempted to progress in this direction. GUEPS are derived from general human factors principles, such as ‘interfaces should exhibit what you see is what you get’. This concept can be refined into statements of cause and effect, and from there to a set of state constraints which describe how the interface can and cannot behave, for example, taken from the WYSIWYG principle:

- Any operation provided by the system will have an equivalent effect on the screen as in the data
- No hidden side effects may occur (data may not be modified without a corresponding screen display to inform the user)
- It is always possible to generate a visible description of the data which is available to the current toolset

Once the general statements have been refined to express permissible states for the system and its interface, and conditions for transitions between these states, the principles can be formalised in an abstract model. While GUEPS have the potential to improve interface design, their application would need to be evaluated in practice. Also because GUEPS are dependent on improvements in psychological knowledge for the production of general principles which may be formalisable, it is impossible to say how complete, or even sufficient, a set of GUEPS would have to be to ensure a good interface design.

Formal methods in interface design face the same problems as they do in software engineering. Many formalisms become cumbersome, not to say unworkable, with large systems; in addition, formalisation creates a learning problem for the average designer. Hence formal methods have an interface design problem all of their own—how to hide the mathematical formalism from those who wish to use it but do not want the learning burden of a formal language. Formalism of human–computer interaction will probably make slow progress in the short term because of the variability of possible interactions between people and computers, and because of the context dependency of current knowledge. But formalism will continue to be a necessary aim to link interface design with advances in the formal design of software and eventually to specify types of interaction in more rigorous terms. This pre-supposes that we shall eventually succeed in understanding the cognition of interaction.

10.6 Summary

Interface design is part of a larger process of software development, and user involvement in the whole process is essential for good design. Besides user participation in design teams, prototyping approaches can be helpful in obtaining feedback on designs. Designs should be evaluated as early as

possible in the development life cycle and later when implemented. Evaluation can take several forms depending on the evaluators' objectives however, goals for usability should be set including utility, effectiveness, learnability and attitude.

Adaptive interfaces are a potential solution to the mixed user population dilemma, although the consistency of the interface has to be maintained as it adapts. The quest for adaptive interfaces is related to separating all interface software as user interface managers. UIMs can be built independently of applications software for basic handling of interaction but the problem of separability becomes more difficult at the task level. Another quest in interface design has been to apply software engineering principles to interaction and to derive a set of context-independent rules for design. Formal specification of dialogues is difficult because of the variability in users, the context of interaction, and the incomplete knowledge of the psychology of interaction.

Further Reading

For further discussion on human involvement in the design of systems, including humans as an essential part of that design, see Mumford *et al.* (1978) and Checkland (1981). References to the issues of evaluation, adaptation, UIMs and formalisation have been cited in the text; more material can be found in the conference proceedings in the INTERACT, CHI and HCI series.

10.7 Postscript

In a subject which is so new it is probably unwise to speculate about its future, so these remarks will be confined to a few possible directions in which the subject may develop.

Little has been said in this book about the effect of computers on people. The introduction of computerised systems changes our model of the world and the way in which we work in several ways. We may do jobs in a different manner, reflecting changes in our task model; either because current systems force changes upon us, or less frequently, because a computer system allows us to do a new task more creatively. Our interaction with other people is also changed by human-computer interaction. Consider how we exchange messages by telephone, face to face, and by an electronic mail system. Face to face we use facial expressions, gestures and movements to help regulate speech and communication; by telephone none of these is available although we can use intonation of the

voice to help communication. When faced with electronic mail there is nothing but the message itself.

Psychologists and builders of computer-based messaging systems have yet to decide what communication devices should do to help message passing, what types of interaction are possible and what impact computers may make on group behaviour. Human-computer interaction goes beyond psychology into the study of group interaction, that is, sociology. How groups of people interact electronically is currently poorly understood, yet electronic communication is increasing rapidly in many spheres. Computers have and will continue to change our work patterns. The Xerox Corporation introduced the idea of home workers, giving each manager a workstation connected to the office so he or she could work remotely at home. Some authors have anticipated that the office may become redundant as all office communication becomes mediated by computers. On the other hand, people may still need the social stimulus of meeting their friends in the office. Sociologists have yet to answer this question. Speculation on future trends is a nearly boundless topic; many interesting ideas can be found in a good survey by Nickerson (1986) and a series of concept papers in Norman and Draper (1986).

Interface design has progressed through two discernible generations. The first generation used text-based interaction and was either difficult to use or inflexible, such as the familiar menus, form-filling and command language dialogues which still implement many human-computer interfaces. The second generation of direct manipulation interfaces introduced more naturalistic interaction with visual communication. The next generation of interfaces will need to integrate methods of interaction, making considerable use of voice and natural language with advanced graphics. Other interaction media, such as eye movement and gesture, may also be used, and images will be in three dimensions with animation to guide and explain interaction and tasks. A good glimpse of some of the future can be found in Bolt (1984).

Human-computer interaction can be considered to be a close relative of artificial intelligence. Many of the issues are common, such as user modelling, goal-directed processing and knowledge models; and interface design uses some of the practices of artificial intelligence in adaptive interfaces. Within previous chapters there has been no discussion about interfaces to intelligent systems, and their more applied counterparts, expert systems. The reason is that little is yet known about these interfaces. A variety of dialogue styles can be employed from question and answer sessions, typically found in early systems (such as MYCIN— see Rich (1984) for a description of AI systems) to more sophisticated command language dialogues. The problem is more complicated with expert systems because there are two interfaces; an end user interface and the ex-

pert-expert system interface. The latter, in particular, presents considerable challenges in the design of interfaces which allow for the display and editing of complicated knowledge bases. Early styles of interaction left the initiative primarily with the expert computer system; more recently, cooperative expert systems have been developed which have mixed initiative dialogues. Planning and controlling such dialogues presents a considerable challenge.

Human-computer interaction will become more involved with artificial intelligence both in the implementation of more sophisticated UIMs and as a topic within the development of intelligent knowledge-based systems. As HCI progresses more deeply into the psychology of cognition it may also be expected to spread out into group and inter-personal communication within the realms of sociology. Future directions should, however, not obscure the fact that much of the basics of interface design need to be defined and the foundations of the subject underpinned with firm theory. More empirical research is required to resolve contradictory guidelines; more general and predictive theories of cognition are required to model interaction; and design guidelines should be formalised into methods. Formalisation eventually needs to converge with the approach of software engineering to create specifications, and hence designs, which can be verified to ensure that good interfaces are constructed. Clearly there is much work, and plenty of problems, in the present, with many challenges for researchers and designers in the future.

References and Further Reading

The references are organised in two sections; first in chapter order and then a general reference section for further reading. The chapter references are cited in the main text.

Chapter 1

Martin, J (1973). *Design of Man Computer Dialogues*, Prentice-Hall, Englewood Cliffs, NJ.

Chapter 2

Baddeley, A. S. (1979). *Your Memory: A Users Guide*, Pelican, London.
An excellent and readable guide to human memory and associated cognitive phenomena.

Card, S. K., Moran, T. P. and Newell, A. A. (1983). *The Psychology of Human Computer Interaction*, Lawrence Erlbaum Associates, Hillsdale, NJ.
Description of the GOMS model, model human processor and work at Xerox on HCI. Useful intermediate level reading.

Christie, B. and Gardiner, M. M. (1987). *Applying Cognitive Psychology to User Interface Design*, Wiley, London.
An in-depth review of recent research in psychological topics which are relevant to HCI. End of chapter summaries give HCI guidelines derived from psychological research.

Frisby, J. P. (1979). *Seeing: Illusion, Brain and Mind*. Oxford University Press.
A readable and well illustrated discourse on visual perception.

Fry, D. B. (1977). *Homo Loquens*, Cambridge University Press.
General introduction to speech perception.

- Glass, A. L., Holyoak, K. J. and Santa, J. L. (1979). *Cognition*, Addison-Wesley, Reading, MA.
A good textbook covering all aspects of cognition for those who wish to explore the psychological background further. Now in a second edition.
- Hitch, G. J. (1987). 'Working memory', in B. Christie and M. M. Gardiner (eds), *Applying Cognitive Psychology to User Interface Design*, Wiley, New York.
- Johnson-Laird, P. N. (1983). *Mental Models*, Cambridge University Press.
A highly influential text on cognition. Advanced reading for aspiring cognitive scientists.
- Lindsay, P. H. and Norman, D. A. (1977). *Human Information Processing*, Academic Press, London.
An alternative general text on cognition written more explicitly from a 'computational' view. Also in a second edition (1987).
- Marr, D. (1982). *Vision*, Oxford University Press.
Definitive account of vision from Marr's important research. Advanced reading.
- Maslow, A. H. (1987). *Motivation and Personality*, 3rd edition, Harper and Row, New York.
A series of papers on motivation by Maslow and updated by others in the field since the original publication date.
- Miller, G. A. (1956). 'The magical number seven, plus or minus two: some limits on our capacity for processing information', *Psychological Review*, **63**:81-97.
- Newell, A. and Simon, H. (1972). *Human Problem Solving*. Prentice-Hall, Englewood Cliffs, NJ.
- Rumelhart, D. E. and McClelland, J. L. (1987). *Parallel Distributed Processing, Vols 1 and 2*, MIT Press, Cambridge, MA.
Advanced reading on memory and computer modelling.
- Warren, R. M. and Warren, R. P. (1970). 'Auditory illusions and confusions', *Scientific American*, **223**:30-36.

Chapter 3

- Bailey, R. W. (1982). *Human Performance Engineering: A Guide for System Designers*, Prentice-Hall, Englewood Cliffs, NJ.
A comprehensive text covering all ergonomics; good background reading on human factors.
- Damodaran, L., Simpson, A. and Wilson, P. (1980). *Designing Systems for People*, NCC Press/University of Loughborough.
- De Marco, T. (1978). *Structured Systems Analysis and System Specification*, Yourdon Press, New York.
- Gane, C. and Sarson, T. (1979). *Structured Systems Analysis: Tools and Techniques*, Prentice-Hall, Englewood Cliffs, NJ.
- Shackel, B. (ed.) (1974). *Applied Ergonomics Handbook*, Butterworth, Guildford.
Covers ergonomic issues which affect HCI.

Chapter 4

- Card, S. K., Moran, T. P. and Newell, A. (1981). 'The keystroke level model for user performance time with interactive computer systems', *Communications of the ACM*, **23**:396–410.
- Foley, J. D. and van Dam, A. (1982). *Fundamentals of Interactive Computer Graphics*, Addison-Wesley, Reading, MA.
- Johnson, P. (1985). 'Towards a task model of messaging: an example of the application of TAKD to user interface design, in P. Johnson and S. Cook (eds), *People and Computers: Designing the Interface (HCI-85)*, Cambridge University Press.
- Kieras, D. and Polson, P. G. (1985). 'An approach to the formal analysis of user complexity', *Int. J. Man Machine Studies*, **22**:365–394.
- Moran, T. P. (1981). 'The Command Language Grammar: a representation scheme for the user interface of interactive systems', *Int. J. Man Machine Studies*, **15**:3–50.

Reisner, P. (1984). 'Formal grammar as a tool for analysing ease of use: some fundamental concepts, in J. C. Thomas and M. L. Schneider (eds), *Human Factors in Computing Systems*, Ablex, Norwood, NJ.

Shneiderman, B. (1981). 'Multi party grammars and related features for defining interactive systems' *IEEE Transaction on Systems, Machines and Cybernetics*, **12**:148-154.

Shneiderman, B. (1987). *Designing the User Interface*, Addison-Wesley, Reading, MA.

Chapter 5

Kieras, D. and Polson, P. G. (1985). 'An approach to the formal analysis of user complexity', *Int. J. Man Machine Studies*, **22**:365-394.

Chapter 6

Galitz, A. O. (1981). *Handbook of Screen Format Design*. QED Information Sciences, Wellesley, MA.

Huckle, B. (1981). *The Man Machine Interface: Guidelines for the User System Conversation*, Savant Institute, Camforth, Lancs.

Chapter 7

See General references.

Chapter 8

Siegel, S. (1956). *Non Parametric Statistics for the Behavioural Sciences*, McGraw-Hill, New York.

See also General references.

Chapter 9

Schank, R. C. and Abelson, R. P. (1977). *Scripts, Plans, Goals and Understanding*, Lawrence Erlbaum, Newark, NJ.

Shneiderman, B. (1983). 'Direct manipulation: a step beyond programming languages', *IEEE Computer*, **16**(8):57–65.

See also General references.

Chapter 10

Akscyn, R. M. and McCracken, D. L. (1983). *ZOG and the USS Carl Vinson: Lessons in System Development*, Computer Science Department. Report No. CMU CS 84 127, Carnegie Mellon University, USA.

Alty, J. L. (1984). 'Use of path algebras in an interactive adaptive dialogue system', in B. Shackel (ed.), *Proceedings of IFIP conference 'Interact 84'*, Vol. 1, North-Holland, Amsterdam, pp. 321–324.

Bolt, R. A. (1984). *The Human Interface*, Lifetime Learning Series, Wadsworth, London.

Checkland, P. (1981). *Systems Thinking, Systems Practice*, Wiley, New York.

Cockton, G. (1987). 'A new model for separable interactive systems' in H-J. Bullinger and B. Shackel (eds), *Proceedings Interact-87*, North-Holland, Amsterdam, pp. 1033–1040.

Dix, A. and Runciman, C. (1985). 'Abstract models of interactive systems', in P. Johnson and S. Cook (eds), *People and Computers: Designing the Interface (HCI-85)*, Cambridge University Press, pp. 13–22.

Harrison, M. D. and Thimbleby, H. W. (1985). 'Formalising guidelines for the design of interactive systems', in P. Johnson and S. Cook (eds), *People and Computers: Designing the Interface (HCI-85)*, Cambridge University Press, pp. 161–171.

Mumford, E. *et al.* (1978). 'A participative approach to the design of computer systems', *Impact of Science on Society*, **28**(3).

Nickerson, R. S. (1986). *Using Computers: The Human Factors of Information Systems*, MIT Press, Cambridge, MA.

Norman, D. A. and Draper, S. W. (eds) (1986). *User Centered System Design*, Lawrence Erlbaum, Newark, NJ.

- Reisner, P. (1984). 'Formal grammar as a tool for analysing ease of use: some fundamental concepts', in J. C. Thomas and M. L. Schneider (eds), *Human Factors in Computing Systems*, Ablex, Norwood, NJ.
- Rich, E. (1983). *Artificial Intelligence*, McGraw-Hill, New York.
- Robson, C. (1973). *Experimental Design and Statistics in Psychology*, Penguin, Harmondsworth, Middlesex.
- Shackel, B. (1986). 'Ergonomics in design for usability', in M. D. Harrison and A. F. Monk (eds), *People and Computers: Designing for Usability (HCI-86)*, Cambridge University Press, pp. 44-64.
- Sufrin, B. (1986). 'Formal methods and interface design,' in M. D. Harrison and A. F. Monk (eds), *People and Computers: Designing for Usability (HCI-86)*, Cambridge University Press, pp. 23-43.
- Wasserman, A. I. (1984). 'Developing interactive information systems with the user software engineering methodology', in B. Shackel (ed.), *Proceedings of IFIP conference 'Interact 84'*, Vol. 1, North-Holland, Amsterdam, pp. 321-324.

General References and Further Reading

- Bailey, R. W. (1982). *Human Performance Engineering: A Guide for System Designers*, Prentice-Hall, Englewood Cliffs, NJ.
A comprehensive text covering all ergonomics; good background reading on human factors.
- Gaines, B. R. and Shaw, M. L. G. (1984). *The Art of Computer Conversation*, Prentice-Hall, Englewood Cliffs, NJ.
Principles of HCI as a set of proverbs for designers.
- Galitz, A. O. (1981). *Handbook of Screen Format Design*, QED Information Sciences, Wellesley, MA.
Good source of practical screen design guidelines.
- Green, T. R. G., Payne, S. J. and Van der Veer, G. C. (1983). *The Psychology of Computer Use*, Academic Press, London.
Collection of papers on human-computer interaction.
- Huckle, B. (1981). *The Man Machine Interface: Guidelines for the User System Conversation*, Savant Institute, Carnforth, Lancs.
Guidelines for screen design and form filling dialogues.

Monk, A. (ed.) (1985). *Fundamentals of Human Computer Interaction*, Academic Press, London.

A collection of papers on many aspects of human-computer interaction. A good source for additional material on psychological background and interface design.

Rubenstein, R. and Hersh, H. (1985). *The Human Factor: Designing Computer Systems for People*, Digital Press.

Good description of process of interface design illustrated with mini case study; includes guidelines for presentation and dialogue.

Shneiderman, B. (1987). *Designing the User Interface*, Addison-Wesley, Reading, MA.

A complete and up-to-date text on HCI with many references and topics on researchers' agenda.

Sime, M. E. and Coombs, M. J. (1983). *Designing for Human Computer Interaction*, Academic Press, London.

Smith, S. L. and Mosier, J. N. (1984). *Designer Guidelines for the User System Software Interface*, Mitre Corporation Report, Mitre Corporation, Bedford, MA.

An extensive collection of HCI guidelines (500+ and increasing each year). The source, in abridged form, of many of the guidelines in chapters 7-9.

For those wishing to explore the literature further, two journals are recommended:

International Journal of Man Machine Studies
Behaviour and Information Technology

In addition, there are three series of conference papers well worth consulting:

CHI series, *Proceedings of the ACM Conferences on Computer Human Interaction*, ACM Press, 1983 onwards.

HCI series, *Proceedings of the BCS Conferences on Human Computer Interaction*, Cambridge University Press, 1985 onwards.

INTERACT series, *Proceedings of the IEEE/IFIP Conferences on Human Computer Interaction*, North-Holland, Amsterdam, 1984 and 1987

Index

- Abbreviations 116–17, 119, 170,
 see also Command languages
 compression 116, 170
 truncation 116, 170
- Animation 193
- Artificial intelligence 5, 193
- Attention 40, 42–3, 44, 48, 111,
 112, 131
- Backtracking *see* Undo
- Bar codes 135
- Brain 10
 anatomy of 11
- Break points *see* Closure events
- Card, S. K., Moran, T. P. and
 Newell, A. 11, 25, 34, 89
- CCT (Cognitive Complexity
 Theory) 91–4, 95
- CLG (Command Language
 Grammar) 81–8, 95
- Closure events 43, 44, 63, 131
- Codes 117, 170
- Cognition *see* cognitive
 psychology *under* Psychology
- Cognitive overload 44
- Colour 12, 15, 17, 112, 113,
 114–16
 blindness 15, 115
 brightness 13, 115
 guidelines 115
 hue 115
 saturation 115
 guidelines 115, 125
- Command languages 73–4, 141,
 169–74
 design of 173
 grammar 143, 171–2
 guidelines 174
 lexicon 169
- Conceptual dependency 176
- Context (of dialogue,
 interface) 46, 64, 85
- Data
 capture 121–2
 display 137–55
 guidelines 137
 editing 131–3
 entry 72, 119–36
 guidelines 119, 120, 121
 grouping and ordering of 109,
 122, 138, 151
 retrieval 72, 140–4
 guidelines 143
- Databases 28, 35, 140, 141, 178
- De Marco, T. 50, 53
- Decibel 22
- Defaults 102, 119, 120
- Diagrams
 data flow 50–3
 generalised transition
 network 90–4
 network 103–6
 state transition 89, 103
- Dialogue 97–107, 188
 command and
 control 156–80
 design 87, 97–105
 formal specification
 of 190–1
 guidelines 101–2, 157, 158,
 161, 169
- Direct manipulation 166–8, 169,
 189, 193
 principles of 167
- Entities, conceptual entities 82,
 84, 88, 141
- Ergonomics 1, 56
- Error messages 129, 131, 173, 188

- Errors 40, 129, 173
 - mistakes 40, 41
 - slips 40
- Escape 102, 106, 159
- Evaluation 2, 182–6, *see also*
 - Usability
 - analysis of 183
 - data recording for 183
 - goals 184, 185
 - types of 183
- Expert systems 35, 193, 194
 - co-operative 194

- Fatigue 44, 64
- Form-filling dialogues 72–3, 127–33
- Forms design 121–7
 - guidelines 123–7
 - instructions 127
 - layout 122–3, 125–7
 - prompts 127
- Fourier analysis 20
- Function keys 162–4
- Functional decomposition 50, 100

- GIGO (Garbage In Garbage Out) 119
- GOMS (Goals, Operators, model) 34, 41, 85, 89, 90, *see also* Card, S. K., Moran, T. P. and Newell, A.
- Grammar *see* CLG (Command Language Grammar)
 - command languages 87, 88–9, 95
 - language 23, 174–5
- Graphics 108, 144–8, 193
- Graphs 144–8
 - complex 147–8
 - guidelines 144–5
 - histograms 145
 - line graphs 145
 - pie charts 145
 - scattergrams 146
- Group behaviour 193
- GUEPs (Generative User Engineering Principles) 191
- Guidelines *see under* appropriate topic

- Hearing 19–24
- Help 102, 185, 187
- Heuristics 40, 41, 45, 61
- Human factors *see* Ergonomics

- Icons 16, 59, 71–2, 164–6, 167, 168
 - ambiguity of 164, 165
 - design of 164
 - guidelines 166
- Intelligent interface *see* adaptive *under* Interfaces
- Interfaces
 - adaptive 186–7
 - analysis of 49–59
 - design styles 66–75, 78–9, 98–9
 - efficiency of 47, 106, 184
 - justification for design of 3
 - sophistication 77–8
 - support 77–8
 - usability 47, 184, 185
 - utility 46, 47
- Iterative design *see* Prototyping

- Job design 63–4, 192
- Johnson-Laird, P. N. 38

- Kieras, D. and Polson, P. G. 90, 92
- Knowledge 31, 177
 - declarative 31
 - procedural 31
- Knowledge base 177

- Language 23, 74–5, 174–9
 - ambiguity of 175, 177, 179, 193
 - interface design for 178–9
 - guidelines 179
 - lexicon 174, 177
 - phonemes 23, 179
 - pragmatics 23, 176, 177
 - semantics 23, 175
 - syntax 23, 174, 175
- Learning 24, 27, 30–1, 41, 47
- Light 11–14
 - brightness 12–14
 - contrast 12–14
 - luminance 12–14
- Local experts 181, 182

- Memorisation 27–8, 32
 - cue overload 33
 - recall 27, 28, 33

- Memorisation (*cont'd*)
 techniques for 32–34
- Memory 24–34
 access mechanisms 27–30
 associative 28–9, 33
 categories 29, 31, 33
 chunking 26–7, 144
 design guidelines for 27–8, 33–4
 episodic 32, 41
 long-term 24, 28–32
 models of 24–32, 36–9
 short-term 24–8, 31, 46, 63, 69
 auditory 24–7, 41
 cognitive 26–7
 visual 24, 148
 structure of 27, 29–32
 working 27, 38, 39, 90, 92
- Mental models *see* mental *under*
 Models
- Menus 68–9, 133, 158–62, *see*
also design styles *under*
 Interfaces
 guidelines 161
 hierarchy of 159, 160
 options 159, 160
 pull-down/pop-up 161, 190
- Metaphors 58, 79, 166, 168
- Methods 85–6, 91, *see also* CLG
 (Command Language
 Grammar), Structured analysis,
and analysis *under* Task
 user-centred 181–2
 user-participative 181
- MICR (Magnetic Ink Character
 Recognition) 135
- Mode/modeless interaction 163–4
- Models 56
 cognitive 10–11, 57
 conceptual 56, 57, 59, 95, 167
 information processing 24,
 25–6, 28, 42–3
 mental 36–9, 41, 47
 user 47, 56–9, 157
- Modules
 interface 100, 109, 187
 logical 101
 physical 101
- Moran, T. P. 81, 82, 87
- Motivation 43, 59
- Mouse 133, 166
- Mnemonics 32, 116, 168
- Multi-tasking 42, 169
- Natural language *see* Language
- Nerve cells 6–8
- Nerve impulses 8–9
- Neural computing 9–10, 30
- OCR (Optical Character
 Recognition) 135
- OMR (Optical Mark
 Recognition) 133–4
- Operations 84–6, 88
- Optical cortex 16–17
- Parallel distributed processing 30
- Path algebras 190
- Pointing 119, 166, 167, 189, 190
- Presentation 87, 108–18, 188, *see*
also Screen design
- Principles 45, 101–2, 106, 189,
 190, 191
 adaptability 46, 186
 compatibility 46, 47, 58
 consistency 34, 45, 102, 186,
 187
 economy 46, 102
 guidance 46, 102
 predictability 46
 reversibility 46
 structure 34, 46
- Print 15, 139–40, 154
 font 139
 format of 139–40, 154
 point 15, 139
- Problem solving 34–9
 goals of 35, 41
 logical form of 37–8
 models of 34–9
 strategies for 35
- Processors 24, 27–8, 42
 cognitive 24, 27, 28, 42, 43
 motor 25, 42
 sensory 24, 42
- Protocol analysis 183
- Prototyping 97, 98, 182
- Psychology 1, 4, 6–44
 cognitive psychology 11–44, 194
- Question and answer
 dialogues 68, 157, *see also*
 command and control *under*
 Dialogue
- Questionnaires 184

- Reasoning 35–8, 41, 45 *see also*
 - Problem solving
 - backward chaining 35
 - forward chaining 35
 - guidelines for 41
 - inductive 35
 - strategies of 35–8
 - sylogistic 37
- Reisner, P. 88
- Reports 148–54
 - analysis for 150–1
 - guidelines 151, 154
 - types of 150
- Response time 104

- Screen design 108–18, 129–31
 - crowding 109, 137
 - guidelines 110, 115, 116, 137, 138
 - highlighting 111–13, 130
 - messages 116–17, 131, 187
- Scrolling 140
- Semantic networks 29, 31
- Shneiderman, B. 89, 95, 166
- Skills 40–2, 55, 187
 - power law of 40
- Software engineering 5, 191, 194
- Sound 19–23
 - amplitude 20, 22
 - frequency 20, 22, 23
 - harmonics 20, 22
 - pitch 20
- Spatial data management 141
- Speech 19, 20, 23, 179–80
 - generation 23
 - recognition 23–4, 136, 179
- SQL (Structured Query Language) 141, 143
- Stress 44
- Structured analysis 50–3
 - data flow diagrams 50, 52–3
 - structured English 53, 61, 103
- Structured design 100
 - cohesion 100, 152
- Systems
 - documentation 65
 - environment 64–6
 - guidelines 66
 - support 64
- Systems analysis 4, 49–50

- Task 2, 38, 43, 81, 88, 89, 98
 - allocation 61–3
 - analysis 4, 49–53, 58, 100, 101
 - design 38, 43, 59–63, 88
 - fit 47, 92–3
 - models 58, 88–9
 - network 61, 62, 98
 - specification 61, 62
- Text, display of *see* Print
- Trade-off decisions 46, 102, 120, 122
- Training 65, 173
- Transactions 120, 122, 149, 154

- UIMs (User Interface Management systems) 169, 187–90, 194
 - separability 189
- Undo 102, 121, 159
- Usability 47, 184, 185
- User manuals 65–6
- Users
 - analysis of 53, 55–6, 77–8, 110, 122
 - categories 54
 - expert 54, 55, 74, 173, 186
 - naive 54, 68, 75, 158
 - novice 54, 68, 70, 72, 75, 186
 - skilled 54
 - characteristics 53–4, 98, 110, 122
 - modelling 57–8
 - navigation 138, 158
 - terminology 139
 - testing 109, 154, 166
 - views 56–8, 98–9, 100

- Validation 111, 128–9
- VDU displays 14, 109, 111
 - flicker 14
- Vision 11–19
 - acuity 14–15
 - illusions 13, 17–19
 - pre-processing 14–15
 - sensitivity 14
 - threshold 15–17

- WIMP (Windows, Icons, Mouse, Pull-down menus) 166
- Windows 168–9, 190
- WYSIWYG 168, 191

- Z 190