ALAN DIX, JANET FINLAY, GREGORY D. ABOWD, RUSSELL BEALE

HUMAN–COMPUTER INTERACTION

THIRD EDITION

PEARSON
Prentice Hall

Human–Computer Interaction

Human–Computer Interaction

Third Edition

Alan Dix, *Lancaster University*

Janet Finlay, *Leeds Metropolitan University*

Gregory D. Abowd, *Georgia Institute of Technology*

Russell Beale, *University of Birmingham*

PEARSON
Prentice Hall

Harlow, England • London • New York • Boston • San Francisco • Toronto • Sydney • Singapore • Hong Kong
Tokyo • Seoul • Taipei • New Delhi • Cape Town • Madrid • Mexico City • Amsterdam • Munich • Paris • Milan

Pearson Education Limited
Edinburgh Gate
Harlow
Essex CM20 2JE
England

and Associated Companies throughout the world

Visit us on the world wide web at:
www.pearsoned.co.uk

First published 1993
Second edition published 1998
Third edition published 2004

ISBN-13: 978-0-13-046109-4
ISBN-10: 0-13-046109-1

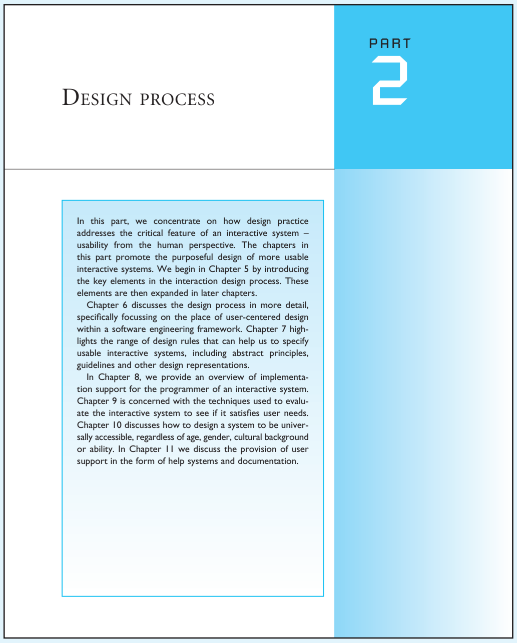
British Library Cataloguing-in-Publication Data
A catalogue record for this book is available from the British Library

10 9 8 7 6 5 4 3
10 09 08 07 06

Typeset in 10/12$^1/_2$pt Minion by 35
Printed and bound by Scotprint, Haddington

Brief contents

Contents



PART 2

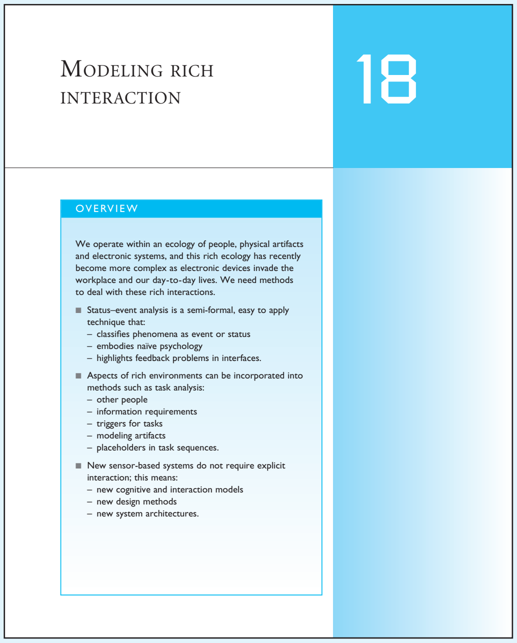
DESIGN PROCESS

In this part, we concentrate on how design practice addresses the critical feature of an interactive system – usability from the human perspective. The chapters in this part promote the purposeful design of more usable interactive systems. We begin in Chapter 5 by introducing the key elements in the interaction design process. These elements are then expanded in later chapters.

Chapter 6 discusses the design process in more detail, specifically focussing on the place of user-centered design within a software engineering framework. Chapter 7 highlights the range of design rules that can help us to specify usable interactive systems, including abstract principles, guidelines and other design representations.

In Chapter 8, we provide an overview of implementation support for the programmer of an interactive system. Chapter 9 is concerned with the techniques used to evaluate the interactive system to see if it satisfies user needs. Chapter 10 discusses how to design a system to be universally accessible, regardless of age, gender, cultural background or ability. In Chapter 11 we discuss the provision of user support in the form of help systems and documentation.

The part structure separates out introductory and more advanced material, with each part opener giving a simple description of what its constituent chapters cover



MODELING RICH INTERACTION

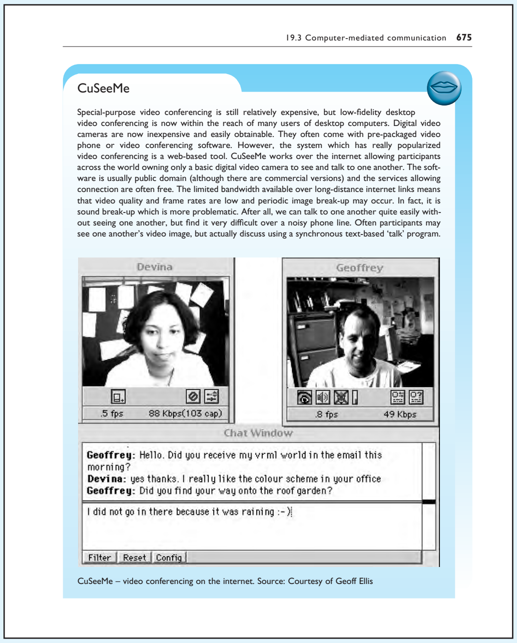
18

OVERVIEW

We operate within an ecology of people, physical artifacts and electronic systems, and this rich ecology has recently become more complex as electronic devices invade the workplace and our day-to-day lives. We need methods to deal with these rich interactions.

- Status–event analysis is a semi-formal, easy to apply technique that:
  - classifies phenomena as event or status
  - embodies naïve psychology
  - highlights feedback problems in interfaces.
- Aspects of rich environments can be incorporated into methods such as task analysis:
  - other people
  - information requirements
  - triggers for tasks
  - modeling artifacts
  - placeholders in task sequences.
- New sensor-based systems do not require explicit interaction; this means:
  - new cognitive and interaction models
  - new design methods
  - new system architectures.

Bullet points at the start of each chapter highlight the core coverage



19.3 Computer-mediated communication 675

CuSeeMe

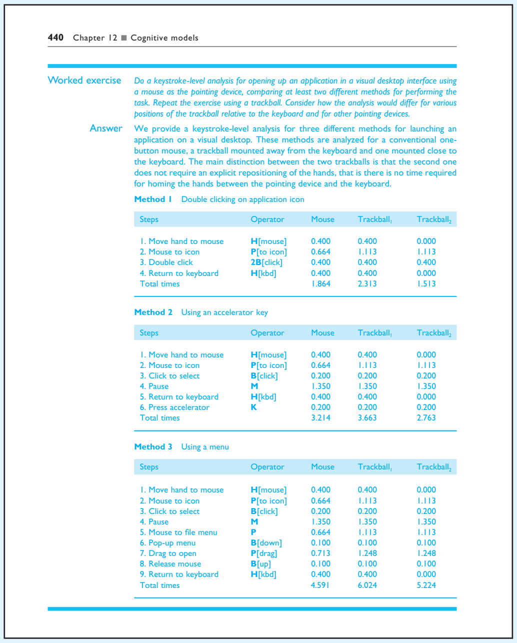
Special-purpose video conferencing is still relatively expensive, but low-fidelity desktop video conferencing is now within the reach of many users of desktop computers. Digital video cameras are now inexpensive and easily obtainable. They often come with pre-packaged video phone or video conferencing software. However, the system which has really popularized video conferencing is a web-based tool. CuSeeMe works over the internet allowing participants across the world owning only a basic digital video camera to see and talk to one another. The software is usually public domain (although there are commercial versions) and the services allowing connection are often free. The limited bandwidth available over long-distance internet links means that video quality and frame rates are low and periodic image break-up may occur. In fact, it is sound break-up which is more problematic. After all, we can talk to one another quite easily without seeing one another, but find it very difficult over a noisy phone line. Often participants may see one another's video image, but actually discuss using a synchronous text-based 'talk' program.

Geoffrey: Hello. Did you receive my vrml world in the email this morning?
Devina: yes thanks. I really like the colour scheme in your office
Geoffrey: Did you find your way onto the roof garden?

I did not go in there because it was raining :-)

CuSeeMe – video conferencing on the internet. Source: Courtesy of Geoff Ellis

Boxed asides contain descriptions of particular tasks or technologies for additional interest, experimentation and discussion



440 Chapter 12 ■ Cognitive models

Worked exercise *Do a keystroke-level analysis for opening up an application in a visual desktop interface using a mouse as the pointing device, comparing at least two different methods for performing the task. Repeat the exercise using a trackball. Consider how the analysis would differ for various positions of the trackball relative to the keyboard and for other pointing devices.*

Answer We provide a keystroke-level analysis for three different methods for launching an application on a visual desktop. These methods are analyzed for a conventional one-button mouse, a trackball mounted away from the keyboard and one mounted close to the keyboard. The main distinction between the two trackballs is that the second one does not require an explicit repositioning of the hands, that is there is no time required for homing the hands between the pointing device and the keyboard.

Method 1 Double clicking on application icon

Steps	Operator	Mouse	Trackball_1	Trackball_2
1. Move hand to mouse	**H**[mouse]	0.400	0.400	0.000
2. Mouse to icon	**P**[to icon]	0.664	1.113	1.113
3. Double click	**2B**[click]	0.400	0.400	0.400
4. Return to keyboard	**H**[kbd]	0.400	0.400	0.000
Total times		1.864	2.313	1.513

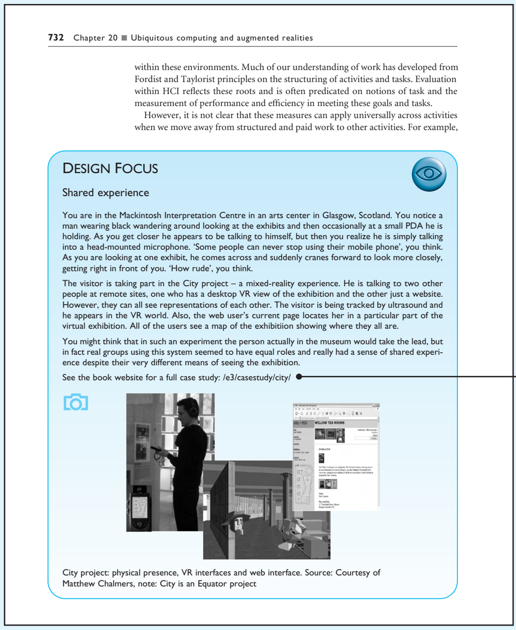
Method 2 Using an accelerator key

Steps	Operator	Mouse	Trackball_1	Trackball_2
1. Move hand to mouse	**H**[mouse]	0.400	0.400	0.000
2. Mouse to icon	**P**[to icon]	0.664	1.113	1.113
3. Click to select	**B**[click]	0.200	0.200	0.200
4. Pause	**M**	1.350	1.350	1.350
5. Return to keyboard	**H**[kbd]	0.400	0.400	0.000
6. Press accelerator	**K**	0.200	0.200	0.200
Total times		3.214	3.663	2.763

Method 3 Using a menu

Steps	Operator	Mouse	Trackball_1	Trackball_2
1. Move hand to mouse	**H**[mouse]	0.400	0.400	0.000
2. Mouse to icon	**P**[to icon]	0.664	1.113	1.113
3. Click to select	**B**[click]	0.200	0.200	0.200
4. Pause	**M**	1.350	1.350	1.350
5. Mouse to file menu	**P**	0.664	1.113	1.113
6. Pop-up menu	**B**[down]	0.100	0.100	0.100
7. Drag to open	**P**[drag]	0.713	1.248	1.248
8. Release mouse	**B**[up]	0.100	0.100	0.100
9. Return to keyboard	**H**[kbd]	0.400	0.400	0.000
Total times		4.591	6.024	5.224

Worked exercises within chapters provide step-by-step guidelines to demonstrate problem-solving techniques



732 Chapter 20 ■ Ubiquitous computing and augmented realities

within these environments. Much of our understanding of work has developed from Fordist and Taylorist principles on the structuring of activities and tasks. Evaluation within HCI reflects these roots and is often predicated on notions of task and the measurement of performance and efficiency in meeting these goals and tasks.

However, it is not clear that these measures can apply universally across activities when we move away from structured and paid work to other activities. For example,

DESIGN FOCUS

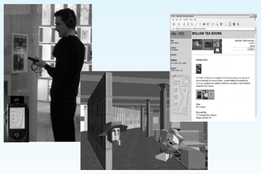
Shared experience

You are in the Mackintosh Interpretation Centre in an arts center in Glasgow, Scotland. You notice a man wearing black wandering around looking at the exhibits and then occasionally at a small PDA he is holding. As you get closer he appears to be talking to himself, but then you realize he is simply talking into a head-mounted microphone. 'Some people can never stop using their mobile phone', you think. As you are looking at one exhibit, he comes across and suddenly cranes forward to look more closely, getting right in front of you. 'How rude', you think.

The visitor is taking part in the City project – a mixed-reality experience. He is talking to two other people at remote sites, one who has a desktop VR view of the exhibition and the other just a website. However, they can all see representations of each other. The visitor is being tracked by ultrasound and he appears in the VR world. Also, the web user's current page locates her in a particular part of the virtual exhibition. All of the users see a map of the exhibitiion showing where they all are.

You might think that in such an experiment the person actually in the museum would take the lead, but in fact real groups using this system seemed to have equal roles and really had a sense of shared experience despite their very different means of seeing the exhibition.

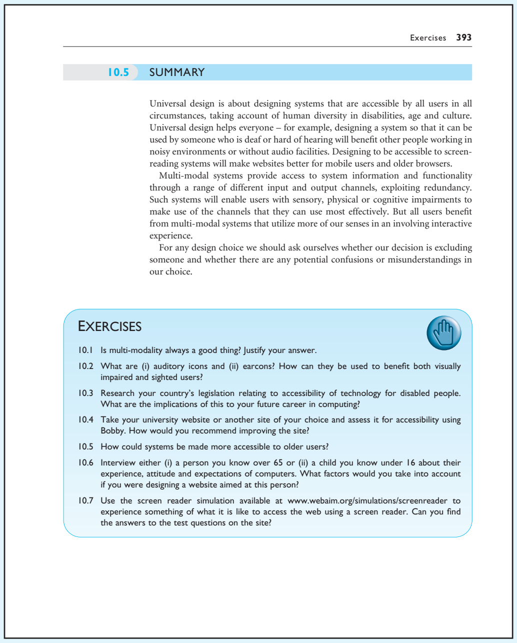
See the book website for a full case study: /e3/casestudy/city/



City project: physical presence, VR interfaces and web interface. Source: Courtesy of Matthew Chalmers, note: City is an Equator project

Frequent links to the book website for further information

Design Focus mini case studies highlight practical applications of HCI concepts



Exercises 393

10.5 SUMMARY

Universal design is about designing systems that are accessible by all users in all circumstances, taking account of human diversity in disabilities, age and culture. Universal design helps everyone – for example, designing a system so that it can be used by someone who is deaf or hard of hearing will benefit other people working in noisy environments or without audio facilities. Designing to be accessible to screen-reading systems will make websites better for mobile users and older browsers.

Multi-modal systems provide access to system information and functionality through a range of different input and output channels, exploiting redundancy. Such systems will enable users with sensory, physical or cognitive impairments to make use of the channels that they can use most effectively. But all users benefit from multi-modal systems that utilize more of our senses in an involving interactive experience.

For any design choice we should ask ourselves whether our decision is excluding someone and whether there are any potential confusions or misunderstandings in our choice.

EXERCISES

10.1 Is multi-modality always a good thing? Justify your answer.

10.2 What are (i) auditory icons and (ii) earcons? How can they be used to benefit both visually impaired and sighted users?

10.3 Research your country's legislation relating to accessibility of technology for disabled people. What are the implications of this to your future career in computing?

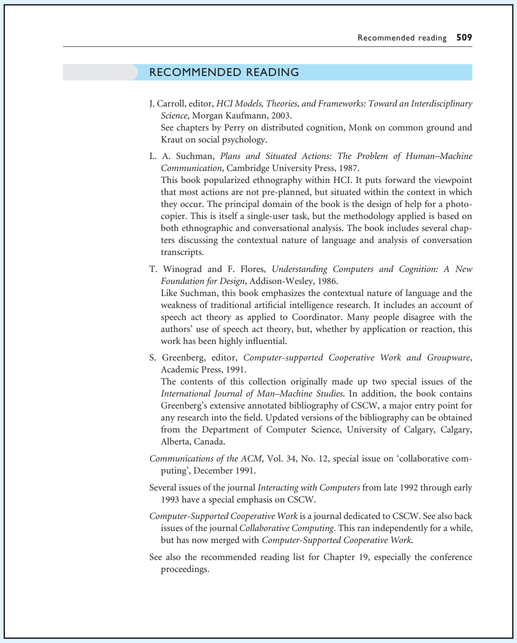
10.4 Take your university website or another site of your choice and assess it for accessibility using Bobby. How would you recommend improving the site?

10.5 How could systems be made more accessible to older users?

10.6 Interview either (i) a person you know over 65 or (ii) a child you know under 16 about their experience, attitude and expectations of computers. What factors would you take into account if you were designing a website aimed at this person?

10.7 Use the screen reader simulation available at www.webaim.org/simulations/screenreader to experience something of what it is like to access the web using a screen reader. Can you find the answers to the test questions on the site?

Chapter summaries reinforce student learning. Exercises at the end of chapters can be used by teachers or individuals to test understanding



Recommended reading 509

RECOMMENDED READING

J. Carroll, editor, *HCI Models, Theories, and Frameworks: Toward an Interdisciplinary Science*, Morgan Kaufmann, 2003.
See chapters by Perry on distributed cognition, Monk on common ground and Kraut on social psychology.

L. A. Suchman, *Plans and Situated Actions: The Problem of Human–Machine Communication*, Cambridge University Press, 1987.
This book popularized ethnography within HCI. It puts forward the viewpoint that most actions are not pre-planned, but situated within the context in which they occur. The principal domain of the book is the design of help for a photocopier. This is itself a single-user task, but the methodology applied is based on both ethnographic and conversational analysis. The book includes several chapters discussing the contextual nature of language and analysis of conversation transcripts.

T. Winograd and F. Flores, *Understanding Computers and Cognition: A New Foundation for Design*, Addison-Wesley, 1986.
Like Suchman, this book emphasizes the contextual nature of language and the weakness of traditional artificial intelligence research. It includes an account of speech act theory as applied to Coordinator. Many people disagree with the authors' use of speech act theory, but, whether by application or reaction, this work has been highly influential.

S. Greenberg, editor, *Computer-supported Cooperative Work and Groupware*, Academic Press, 1991.
The contents of this collection originally made up two special issues of the *International Journal of Man–Machine Studies*. In addition, the book contains Greenberg's extensive annotated bibliography of CSCW, a major entry point for any research into the field. Updated versions of the bibliography can be obtained from the Department of Computer Science, University of Calgary, Calgary, Alberta, Canada.

Communications of the ACM, Vol. 34, No. 12, special issue on 'collaborative computing', December 1991.

Several issues of the journal *Interacting with Computers* from late 1992 through early 1993 have a special emphasis on CSCW.

Computer-Supported Cooperative Work is a journal dedicated to CSCW. See also back issues of the journal *Collaborative Computing*. This ran independently for a while, but has now merged with *Computer-Supported Cooperative Work*.

See also the recommended reading list for Chapter 19, especially the conference proceedings.

Annotated further reading encourages readers to research topics in depth

Foreword

Human–computer interaction is a difficult endeavor with glorious rewards. Designing interactive computer systems to be effective, efficient, easy, and enjoyable to use is important, so that people and society may realize the benefits of computation-based devices. The subtle weave of constraints and their trade-offs – human, machine, algorithmic, task, social, aesthetic, and economic – generates the difficulty. The reward is the creation of digital libraries where scholars can find and turn the pages of virtual medieval manuscripts thousands of miles away; medical instruments that allow a surgical team to conceptualize, locate, and monitor a complex neurosurgical operation; virtual worlds for entertainment and social interaction, responsive and efficient government services, from online license renewal to the analysis of parliamentary testimony; or smart telephones that know where they are and understand limited speech. Interaction designers create interaction in virtual worlds and embed interaction in physical worlds.

Human–computer interaction is a specialty in many fields, and is therefore multidisciplinary, but it has an intrinsic relationship as a subfield to computer science. Most interactive computing systems are for some human purpose and interact with humans in human contexts. The notion that computer science is the study of algorithms has virtue as an attempt to bring foundational rigor, but can lead to ignoring constraints foundational to the design of successful interactive computer systems. A lesson repeatedly learned in engineering is that a major source of failure is the narrow optimization of a design that does not take sufficient account of contextual factors. Human users and their contexts are major components of the design problem that cannot be wished away simply because they are complex to address. In fact, that largest part of program code in most interactive systems deals with user interaction. Inadequate attention to users and task context not only leads to bad user interfaces, it puts entire systems at risk.

The problem is how to take into account the human and contextual part of a system with anything like the rigor with which other parts of the system are understood and designed – how to go beyond fuzzy platitudes like ‘know the user’ that are true, but do not give a method for doing or a test for having done. This is difficult to do, but inescapable, and, in fact, capable of progress. Over the years, the need to take into account human aspects of technical systems has led to the creation of new fields of study: applied psychology, industrial engineering, ergonomics, human factors, man–machine systems. Human–computer interaction is the latest of these, more complex in some ways because of the breadth of user populations and applications, the reach into cognitive and social constraints, and the emphasis on interaction. The experiences with other human-technical disciplines lead to a set of conclusions about how a discipline of human–computer interaction should be organized if it is to be successful.

First, design is where the action is. An effective discipline of human–computer interaction cannot be based largely on 'usability analysis', important though that may be. Usability analysis happens too late; there are too few degrees of freedom; and most importantly, it is not generative. Design thrives on understanding constraints, on insight into the design space, and on deep knowledge of the materials of the design, that is, the user, the task, and the machine. The classic landmark designs in human–computer interaction, such as the Xerox Star and the Apple Lisa/Macintosh, were not created from usability analysis (although usability analysis had important roles), but by generative principles for their designs by user interface designers who had control of the design and implementation.

Second, although the notion of 'user-centered design' gets much press, we should really be emphasizing 'task-centered design'. Understanding the purpose and context of a system is key to allocating functions between people and machines and to designing their interaction. It is only in deciding what a human–machine system should do and the constraints on this goal that the human and technical issues can be resolved. The need for task-centered design brings forward the need for methods of task analysis as a central part of system design.

Third, human–computer interaction needs to be structured to include both analytic and implementation methods together in the same discipline and taught together as part of the core. Practitioners of the discipline who can only evaluate, but not design and build are under a handicap. Builders who cannot reason analytically about the systems they build or who do not understand the human information processing or social contexts of their designs are under a handicap. Of course, there will be specialists in one or another part of human–computer interaction, but for there to be a successful field, there must be a common core.

Finally, what makes a discipline is a set of methods for doing something. A field must have results that can be taught and used by people other than their originators to do something. Historically, a field naturally evolves from a set of point results to a set of techniques to a set of facts, general abstractions, methods, and theories. In fact, for a field to be cumulative, there must be compaction of knowledge by crunching the results down into methods and theories; otherwise the field becomes fad-driven and a collection of an almost unteachably large set of weak results. The most useful methods and theories are generative theories: from some task analysis it is possible to compute some insightful property that constrains the design space of a system. In a formula: task analysis, approximation, and calculation. For example, we can predict that if a graphics system cannot update the display faster than 10 times/second then the illusion of animation will begin to break down. This constraint worked backwards has architectural implications for how to guarantee the needed display rate under variable computational load. It can be designed against.

This textbook, by Alan Dix, Janet Finlay, Gregory Abowd, and Russell Beale, represents how far human–computer interaction has come in developing and organizing technical results for the design and understanding of interactive systems. Remarkably, by the light of their text, it is pretty far, satisfying all the just-enumerated conclusions. This book makes an argument that by now there are many teachable results in human–computer interaction by weight alone! It makes an argument that these results form a cumulative discipline by its structure, with sections that organize the results systematically, characterizing human, machine, interaction, and the design process. There are analytic models, but also code implementation examples. It is no surprise that methods of task analysis play a prominent role in the text as do theories to help in the design of the interaction. Usability evaluation methods are integrated in their proper niche within the larger framework.

In short, the codification of the field of human–computer interaction in this text is now starting to look like other subfields of computer science. Students by studying the text can learn how to understand and build interactive systems. Human–computer interaction as represented by the text fits together with other parts of computer science. Moreover, human–computer interaction as presented is a challenge problem for advancing theory in cognitive science, design, business, or social-technical systems. Given where the field was just a few short years ago, the creation of this text is a monumental achievement. The way is open to reap the glorious rewards of interactive systems through a markedly less difficult endeavor, both for designer and for user.

Stuart K. Card
Palo Alto Research Center, Palo Alto, California

Preface to the third edition

It is ten years since the first edition of this book was published and much has changed. Ubiquitous computing and rich sensor-filled environments are finding their way out of the laboratory, not just into films and fiction, but also into our workplaces and homes. Now the computer really has broken its bounds of plastic and glass: we live in networked societies where personal computing devices from mobile phones to smart cards fill our pockets, and electronic devices surround us at home and at work. The web too has grown from a largely academic network into the hub of business and everyday lives. As the distinctions between physical and digital, work and leisure start to break down, human–computer interaction is also radically changing.

We have tried to capture some of the excitement of these changes in this revised edition, including issues of physical devices in Chapters 2 and 3, discussion of web interfaces in Chapter 21, ubiquitous computing in Chapters 4 and 20, and new models and paradigms for interaction in these new environments in Chapters 17 and 18. We have reflected aspects of the shift in use of technology from work to leisure in the analysis of user experience in Chapter 3, and in several of the boxed examples and case studies in the text. This new edition of *Human–Computer Interaction* is not just tracking these changes but looking ahead at emerging areas.

However, it is also rooted in strong principles and models that are not dependent on the passing technologies of the day. We are excited both by the challenges of the new and by the established foundations, as it is these foundations that will be the means by which today's students understand tomorrow's technology. So we make no apology for continuing the focus of previous editions on the theoretical and conceptual models that underpin our discipline. As the use of technology has changed, these models have expanded. In particular, the insular individual focus of early work is increasingly giving way to include the social and physical context. This is reflected in the expanded treatment of social and organizational analysis, including ethnography, in Chapter 13, and the analysis of artifacts in the physical environment in Chapter 18.

STRUCTURE

The structure of the new edition has been completely revised. This in part reflects the growth of the area: ten years ago HCI was as often as not a minority optional subject, and the original edition was written to capture the core material for a standard course. Today HCI is much expanded: some areas (like CSCW) are fully fledged disciplines in their own right, and HCI is studied from a range of perspectives and at different levels of detail. We have therefore separated basic material suitable for introductory courses into the first two parts, including a new chapter on interaction design, which adds new material on scenarios and navigation design and provides an overview suitable for a first course. In addition, we have included a new chapter on universal design, to reflect the growing emphasis on design that is inclusive of all, regardless of ability, age or cultural background. More advanced material focussing on different HCI models and theories is presented in Part 3, with extended coverage of social and contextual models and rich interaction. It is intended that these sections will be suitable for more advanced HCI courses at undergraduate and postgraduate level, as well as for researchers new to the field. Detailed coverage of the particular domains of web applications, ubiquitous computing and CSCW is given in Part 4.

New to this edition is a full color plate section. Images flagged with a camera icon in the text can be found in color in the plate section.

WEBSITE AND SUPPORT MATERIALS

We have always believed that support materials are an essential part of a textbook of this kind. These are designed to supplement and enhance the printed book – physical and digital integration in practice. Since the first edition we have had exercises, mini-case studies and presentation slides for all chapters available electronically. For the second edition these were incorporated into a website including links and an online search facility that acts as an exhaustive index to the book and mini-encyclopedia of HCI. For visually disabled readers, access to a full online electronic text has also been available. The website is continuing to develop, and for the third edition provides all these features plus more, including WAP search, multi-choice questions, and extended case study material (see also color plate section). We will use the book website to bring you new exercises, information and other things, so do visit us at www.hcibook.com (also available via www.booksites.net/dix). Throughout the book you will find shorthand web references of the form /e3/a-page-url/. Just prepend http://www.hcibook.com to find further information. To assist users of the second edition, a mapping between the structures of the old and new editions is available on the web at: http://www.hcibook.com/e3/contents/map2e/

STYLISTIC CONVENTION

As with all books, we have had to make some global decisions regarding style and terminology. Specifically, in a book in which the central characters are 'the user' and 'the designer', it is difficult to avoid the singular pronoun. We therefore use the pronoun 'he' when discussing the user and 'she' when referring to the designer. In other cases we use 'she' as a generic term. This should not be taken to imply anything about the composition of any actual population.

Similarly, we have adopted the convention of referring to the field of 'Human–Computer Interaction' and the notion of 'human–computer interaction'. In many cases we will also use the abbreviation HCI.

ACKNOWLEDGEMENTS

In a book of this size, written by multiple authors, there will always be myriad people behind the scenes who have aided, supported and abetted our efforts. We would like to thank all those who provided information, pictures and software that have enhanced the quality of the final product. In particular, we are indebted to Wendy Mackay for the photograph of EVA; Wendy Hall and her colleagues at the University of Southampton for the screen shot of Microcosm; Saul Greenberg for the reactive keyboard; Alistair Edwards for Soundtrack; Christina Engelbart for the photographs of the early chord keyset and mouse; Geoff Ellis for the screen shot of Devina and himself using CuSeeMe; Steve Benford for images of the Internet Foyer; and Tony Renshaw who provided photographs of the eye tracking equipment. Thanks too to Simon Shum for information on design rationale, Robert Ward who gave us material on psycho-physiology, and Elizabeth Mynatt and Tom Rodden who worked with Gregory on material adapted in Chapter 20. Several of the boxed case studies are based on the work of multi-institution projects, and we are grateful to all those from the project teams of CASCO, thePooch SMART-ITS, TOWER, AVATAR-Conference and TEAM-HOS for boxes and case studies based on their work; and also to the EQUATOR project from which we drew material for the boxes on cultural probes, 'Ambient Wood' and 'City'. We would also like to thank all the reviewers and survey respondents whose feedback helped us to select our subject matter and improve our coverage; and our colleagues at our respective institutions and beyond who offered insight, encouragement and tolerance throughout the revision. We are indebted to all those who have contributed to the production process at Pearson Education and elsewhere, especially Keith Mansfield, Anita Atkinson, Lynette Miller, Sheila Chatten and Robert Chaundy.

Personal thanks must go to Fiona, Esther, Miriam, Rachel, Tina, Meghan, Aidan and Blaise, who have all endured 'The Book' well beyond the call of duty and over many years, and Bruno and ‘the girls’ who continue to make their own inimitable contribution.

Finally we all owe huge thanks to Fiona for her continued deep personal support and for tireless proofreading, checking of figures, and keeping us all moving. We would never have got beyond the first edition without her.

The efforts of all of these have meant that the book is better than it would otherwise have been. Where it could still be better, we take full responsibility.

PUBLISHER'S ACKNOWLEDGEMENTS

We are grateful to the following for permission to reproduce copyright material:

Figure p. 2, Figures 3.14, 3.15, 3.16 and 5.13 and Exercise 8.4 screen shots reprinted by permission from Apple Computer, Inc.; Figure 2.11 reprinted by permission of Keith Cheverst; Figure 3.13 from The WebBook and Web Forager: An information workspace for the world-wide web in *CHI Conference Proceedings*, © 1996 ACM, Inc., reprinted by permission (Card, S. K., Robertson, G. G. and York, W. 1996); Figures 3.9, 3.19, 5.5, Chapter 14, Design Focus: Looking real – Avatar Conference screen shots, Figures 21.3, 21.10, 21.11 screen shot frames reprinted by permission from Microsoft Corporation; Tables 6.2 and 6.3 adapted from Usability engineering: our experience and evolution in *Handbook for Human–Computer Interaction* edited by M. Helander, Copyright 1988, with permission from Elsevier (Whiteside, J., Bennett, J. and Hotzblatt, K. 1988); Figure 7.1 adapted from The alternate reality kit – an animated environment for creating interactive simulations in *Proceedings of Workshop on Visual Languages*, © 1986 IEEE, reprinted by permission of IEEE (Smith, R. B. 1986); Figure 7.2 from Guidelines for designing user interface software in *MITRE Corporation Report MTR-9420*, reprinted by permission of The MITRE Corporation (Smith, S. L. and Mosier, J. N. 1986); Figure 7.3 reprinted by permission of Jenifer Tidwell; Figures 8.6 and 8.9 from *Xview Programming Manual*, Volume 7 of *The X Window System*, reprinted by permission of O'Reilly and Associates, Inc. (Heller, D. 1990); Figure 9.8 screen shot reprinted by permission of Dr. R. D. Ward; Figure 10.2 after Earcons and icons: their structure and common design principles in *Human-Computer Interaction*, 4(1), published and reprinted by permission of Lawrence Erlbaum Associates, Inc. (Blattner, M., Sumikawa, D. and Greenberg, R. 1989); Figure 10.5 reprinted by permission of Alistair D. N. Edwards; Figure 10.7 reprinted by permission of Saul Greenberg; Figure 11.2 screen shot reprinted by permission of Macromedia, Inc.; Table 12.1 adapted from *The Psychology of Human Computer Interaction*, published and reprinted by permission of Lawrence Erlbaum Associates, Inc. (Card, S. K., Moran, T. P. and Newell, A. 1983); Table 12.2 after Table in A comparison of input devices in elemental pointing and dragging tasks in *Reaching through technology – CHI'91 Conference Proceedings, Human Factors in Computing Systems*, April, edited by S. P. Robertson, G. M. Olson and J. S. Olson, © 1991 ACM, Inc., reprinted by permission (Mackenzie,

I. S., Sellen, A. and Buxton, W. 1991); Figure 14.1 from *Understanding Computers and Cognition: A New Foundation for Design*, published by Addison-Wesley, reprinted by permission of Pearson Education, Inc. (Winograd, T. and Flores, F. 1986); Figure 14.5 from *Theories of multi-party interaction. Technical report*, Social and Computer Sciences Research Group, University of Surrey and Queen Mary and Westfield Colleges, University of London, reprinted by permission of Nigel Gilbert (Hewitt, B., Gilbert, N., Jirotka, M. and Wilbur, S. 1990); Figure 14.6 from *Dialogue processes in computer-mediated communication: a study of letters in the com system. Technical report,* Linköping Studies in Arts and Sciences, reprinted by permission of Kerstin Severinson Eklundh (Eklundh, K. S. 1986); Chapter 14, Design Focus: Looking real – Avatar Conference, screen shots reprinted by permission of AVATAR-Conference project team; Figure 16.17 screen shot reprinted by permission of Harold Thimbleby; Figure 17.5 based on Verifying the behaviour of virtual world objects in *DSV-IS 2000 Interactive Systems: Design, Specification and Verification.* LNCS 1946, edited by P. Palanque and F. Paternò, published and reprinted by permission of Spinger-Verlag GmbH & Co. KG (Willans, J. S. and Harrison, M. D. 2001); Figure 18.4 icons reprinted by permission of Fabio Paternò; Chapter 19, p.675 CuSeeMe screen shot reprinted by permission of Geoff Ellis; Chapter 19, Design Focus: TOWER – workspace awareness, screen shots reprinted by permission of Wolfgang Prinz; Figure 20.1 reprinted by permission of Mitsubishi Electric Research Laboratories, Inc.; Figure 20.4 (right) reprinted by permission of Sony Computer Science Laboratories, Inc; Figure 20.9 from Cone trees. Animated 3d visualisation of hierarchical information in *Proceedings of the CH'91 Conference of Human Factors in Computing Systems*, © 1991 ACM, Inc., reprinted by permission (Robertson, G. G., Card, S. K., and Mackinlay, J. D. 1991); Figure 20.10 from Lifelines: visualising personal histories in *Proceedings of CH'96*, © 1996 ACM, Inc., reprinted by permission (Plaisant, C., Milash, B., Rose, A., Widoff, S. and Shneiderman, B. 1996); Figure 20.11 from Browsing anatomical image databases: a case study of the Visible Human in *CH'96 Conference Companion*, © 1996 ACM, Inc., reprinted by permission (North, C. and Korn, F. 1996); Figure 20.12 from Externalising abstract mathematical models in *Proceedings of CH'96*, © 1996 ACM, Inc., reprinted by permission (Tweedie, L., Spence, R., Dawkes, H. and Su, H. 1996); Figure 21.2 from The impact of Utility and Time on Distributed Information Retrieval in *People and Computers XII: Proceedings of HCI'97*, edited by H. Thimbleby, B. O'Conaill and P. Thomas, published and reprinted by permission of Spinger-Verlag GmbH & Co. KG (Johnson, C. W. 1997); Figure 21.4 screen shot reprinted by permission of the Departments of Electronics and Computer Science and History at the University of Southampton; Figure 21.6 Netscape browser window © 2002 Netscape Communications Corporation. Used with permission. Netscape has not authorized, sponsored, endorsed, or approved this publication and is not responsible for its content.

We are grateful to the following for permission to reproduce photographs:

Chapter 1, p. 50, Popperfoto.com; Chapter 2, p. 65, PCD Maltron Ltd; Figure 2.2 Electrolux; Figures 2.6 and 19.6 photos courtesy of Douglas Engelbart and Bootstrap Institute; Figure 2.8 (left) British Sky Broadcasting Limited; Figure 2.13 (bottom right) Sony (UK) Ltd; Chapter 2, Design Focus: Feeling the Road, BMW AG; Chapter 2, Design Focus: Smart-Its – making using sensors easy, Hans Gellersen; Figures 4.1 (right) and 20.2 (left) Palo Alto Research Center; Figure 4.2 and 20.3 (left) François Guimbretière; Figure 4.3 (bottom left) Franklin Electronic Publishers; Figure 5.2 (top plate and middle plate) Kingston Museum and Heritage Service, (bottom plate) V&A Images, The Victoria and Albert Museum, London; Chapter 5, Design Focus: Cultural probes, William W. Gaver, Anthony Boucher, Sarah Pennington and Brendan Walker, Equator IRC, Royal College of Art; Chapter 6, p. 245, from The 1984 Olympic Message System: a text of behavioural principle of system design in *Communications of the ACM*, 30(9), © 1987 ACM, Inc., reprinted by permission (Gould, J. D., Boies, S. J., Levy, S., Richards, J. T. and Schoonard, J. 1987); Figures 9.5 and 9.6 J. A. Renshaw; Figure 9.7 Dr. R. D. Ward; Figure 10.3 SensAble Technologies; Chapter 13, Design Focus: Tomorrow's hospital – using participatory design, Professor J. Artur Vale Serrano; Chapter 18, p. 650, Michael Beigl; Chapter 19, p. 678, Steve Benford, The Mixed Reality Laboratory, University of Nottingham; Chapter 19, Design Focus: SMS in action, Mark Rouncefield; Figure 20.2 (right) Ken Hinckley; Figure 20.3 (right) MIT Media Lab; Figure 20.4 (left) from Interacting with paper on the digital desk in *Communications of the ACM*, 36(7), © 1993 ACM, Inc., reprinted by permission (Wellner, P. 1993); Chapter 20, p. 726, Peter Phillips; Chapter 20, Design Focus: Ambient wood – augmenting the physical, Yvonne Rogers; Chapter 20, Design Focus: Shared experience, Matthew Chalmers.

We are grateful to the following for permission to reproduce text extracts:

Pearson Education, Inc. Publishing as Pearson Addison Wesley for an extract adapted from *Designing the User Interface: Strategies for Effective Human–Computer Interaction 3/e* by B. Shneiderman © 1998, Pearson Education, Inc; Perseus Books Group for an extract adapted from *The Design of Everyday Things* by D. Norman, 1998; and Wiley Publishing, Inc. for extracts adapted from 'Heuristic Evaluation' by Jakob Nielson and Robert L. Mack published in *Usability Inspection Methods* © 1994 Wiley Publishing, Inc.; IEEE for permission to base chapter 20 on 'The human experience' by Gregory Abowd, Elizabeth Mynatt and Tom Rodden which appeared in *IEEE Pervasive Computing Magazine*, Special Inaugural Issue on Reaching for Weiser's Vision, Vol. 1, Issue 1, pp. 48–58, Jan–March 2002. © 2002 IEEE.

In some instances we have been unable to trace the owners of copyright material, and we would appreciate any information that would enable us to do so.

Introduction

WHY HUMAN–COMPUTER INTERACTION?

In the first edition of this book we wrote the following:

> This is the authors' second attempt at writing this introduction. Our first attempt fell victim to a design quirk coupled with an innocent, though weary and less than attentive, user. The word-processing package we originally used to write this introduction is menu based. Menu items are grouped to reflect their function. The 'save' and 'delete' options, both of which are correctly classified as file-level operations, are consequently adjacent items in the menu. With a cursor controlled by a trackball it is all too easy for the hand to slip, inadvertently selecting delete instead of save. Of course, the delete option, being well thought out, pops up a confirmation box allowing the user to cancel a mistaken command. Unfortunately, the save option produces a very similar confirmation box – it was only as we hit the 'Confirm' button that we noticed the word 'delete' at the top . . .

Happily this word processor no longer has a delete option in its menu, but unfortunately, similar problems to this are still an all too common occurrence. Errors such as these, resulting from poor design choices, happen every day. Perhaps they are not catastrophic: after all nobody's life is endangered nor is there environmental damage (unless the designer happens to be nearby or you break something in frustration!). However, when you lose several hours' work with no written notes or backup and a publisher's deadline already a week past, 'catastrophe' is certainly the word that springs to mind.

Why is it then that when computers are marketed as 'user friendly' and 'easy to use', simple mistakes like this can still occur? Did the designer of the word processor actually try to use it with the trackball, or was it just that she was so expert with the system that the mistake never arose? We hazard a guess that no one tried to use it when tired and under pressure. But these criticisms are not levied only on the designers of traditional computer software. More and more, our everyday lives involve programmed devices that do not sit on our desk, and these devices are just as unusable. Exactly how many VCR designers understand the universal difficulty people have trying to set their machines to record a television program? Do car radio designers actually think it is safe to use so many knobs and displays that the driver has to divert attention away from the road completely in order to tune the radio or adjust the volume?

Computers and related devices have to be designed with an understanding that people with specific tasks in mind will want to use them in a way that is seamless with respect to their everyday work. To do this, those who design these systems need to know how to think in terms of the eventual users' tasks and how to translate that knowledge into an executable system. But there is a problem with trying to teach the notion of designing computers for people. All designers *are* people and, most probably, they are users as well. Isn't it therefore intuitive to design for the user? Why does it need to be taught when we all know what a good interface looks like? As a result, the study of human–computer interaction (HCI) tends to come late in the designer's training, if at all. The scenario with which we started shows that this is a mistaken view; it is not at all intuitive or easy to design consistent, robust systems

DESIGN FOCUS

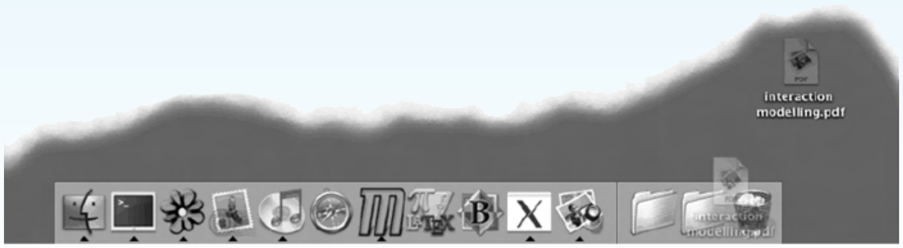
Things don't change

It would be nice to think that problems like those described at the start of the Introduction would never happen now. Think again! Look at the MacOS X 'dock' below. It is a fast launch point for applications; folders and files can be dragged there for instant access; and also, at the right-hand side, there sits the trash can. Imagine what happens as you try to drag a file into one of the folders. If your finger accidentally slips whilst the icon is over the trash can – oops!

Happily this is not quite as easy in reality as it looks in the screen shot, since the icons in the dock constantly move around as you try to drag a file into it. This is to make room for the file in case you want to place it in the dock. However, it means you have to concentrate very hard when dragging a file over the dock. We assume this is not a deliberate feature, but it does have the beneficial side effect that users are less likely to throw away a file by accident – whew!

In fact it is quite fun to watch a new user trying to throw away a file. The trash can keeps moving as if it didn't want the file in it. Experienced users evolve coping strategies. One user always drags files into the trash from the right-hand side as then the icons in the dock don't move around. So two lessons:

- designs don't always get better
- but at least users are clever.



Screen shot reprinted by permission from Apple Computer, Inc.

that will cope with all manner of user carelessness. The interface is not something that can be plugged in at the last minute; its design should be developed integrally with the rest of the system. It should not just present a 'pretty face', but should support the tasks that people actually want to do, and forgive the careless mistakes. We therefore need to consider how HCI fits into the design process.

Designing usable systems is not simply a matter of altruism towards the eventual user, or even marketing; it is increasingly a matter of law. National health and safety standards constrain employers to provide their workforce with usable computer systems: not just safe but *usable*. For example, EC Directive 90/270/EEC, which has been incorporated into member countries' legislation, requires employers to ensure the following when designing, selecting, commissioning or modifying software:

- that it is suitable for the task
- that it is easy to use and, where appropriate, adaptable to the user's knowledge and experience
- that it provides feedback on performance
- that it displays information in a format and at a pace that is adapted to the user
- that it conforms to the 'principles of software ergonomics'.

Designers and employers can no longer afford to ignore the user.

WHAT IS HCI?

The term *human–computer interaction* has only been in widespread use since the early 1980s, but has its roots in more established disciplines. Systematic study of human performance began in earnest at the beginning of the last century in factories, with an emphasis on manual tasks. The Second World War provided the impetus for studying the interaction between humans and machines, as each side strove to produce more effective weapons systems. This led to a wave of interest in the area among researchers, and the formation of the Ergonomics Research Society in 1949. Traditionally, ergonomists have been concerned primarily with the physical characteristics of machines and systems, and how these affect user performance. Human Factors incorporates these issues, and more cognitive issues as well. The terms are often used interchangeably, with Ergonomics being the preferred term in the United Kingdom and Human Factors in the English-speaking parts of North America. Both of these disciplines are concerned with user performance in the context of any system, whether computer, mechanical or manual. As computer use became more widespread, an increasing number of researchers specialized in studying the interaction between people and computers, concerning themselves with the physical, psychological and theoretical aspects of this process. This research originally went under the name *man–machine interaction*, but this became *human–computer interaction* in recognition of the particular interest in computers and the composition of the user population!

Another strand of research that has influenced the development of HCI is information science and technology. Again the former is an old discipline, pre-dating the introduction of technology, and is concerned with the management and manipulation of information within an organization. The introduction of technology has had a profound effect on the way that information can be stored, accessed and utilized and, consequently, a significant effect on the organization and work environment. Systems analysis has traditionally concerned itself with the influence of technology in the workplace, and fitting the technology to the requirements and constraints of the job. These issues are also the concern of HCI.

HCI draws on many disciplines, as we shall see, but it is in computer science and systems design that it must be accepted as a central concern. For all the other disciplines it can be a specialism, albeit one that provides crucial input; for systems design it is an essential part of the design process. From this perspective, HCI involves the design, implementation and evaluation of interactive systems in the context of the user's task and work.

However, when we talk about human–computer interaction, we do not necessarily envisage a single user with a desktop computer. By *user* we may mean an individual user, a group of users working together, or a sequence of users in an organization, each dealing with some part of the task or process. The user is whoever is trying to get the job done using the technology. By *computer* we mean any technology ranging from the general desktop computer to a large-scale computer system, a process control system or an embedded system. The system may include non-computerized parts, including other people. By *interaction* we mean any communication between a user and computer, be it direct or indirect. Direct interaction involves a dialog with feedback and control throughout performance of the task. Indirect interaction may involve batch processing or intelligent sensors controlling the environment. The important thing is that the user is interacting with the computer in order to accomplish something.

WHO IS INVOLVED IN HCI?

HCI is undoubtedly a multi-disciplinary subject. The ideal designer of an interactive system would have expertise in a range of topics: psychology and cognitive science to give her knowledge of the user's perceptual, cognitive and problem-solving skills; ergonomics for the user's physical capabilities; sociology to help her understand the wider context of the interaction; computer science and engineering to be able to build the necessary technology; business to be able to market it; graphic design to produce an effective interface presentation; technical writing to produce the manuals, and so it goes on. There is obviously too much expertise here to be held by one person (or indeed four!), perhaps even too much for the average design team. Indeed, although HCI is recognized as an interdisciplinary subject, in practice people tend to take a strong stance on one side or another. However, it is not possible to design effective interactive systems from one discipline in isolation. Input is needed from all sides. For example, a beautifully designed graphic display may be unusable if it ignores dialog constraints or the psychological limitations of the user.

In this book we want to encourage the multi-disciplinary view of HCI but we too have our 'stance', as computer scientists. We are interested in answering a particular question. How do principles and methods from each of these contributing disciplines in HCI help us to design better systems? In this we must be pragmatists rather than theorists: we want to know how to apply the theory to the problem rather than just acquire a deep understanding of the theory. Our goal, then, is to be multi-disciplinary but practical. We concentrate particularly on computer science, psychology and cognitive science as core subjects, and on their application to design; other disciplines are consulted to provide input where relevant.

THEORY AND HCI

Unfortunately for us, there is no general and unified theory of HCI that we can present. Indeed, it may be impossible ever to derive one; it is certainly out of our reach today. However, there is an underlying principle that forms the basis of our own views on HCI, and it is captured in our claim that people use computers to accomplish work. This outlines the three major issues of concern: the people, the computers and the tasks that are performed. The system must support the user's task, which gives us a fourth focus, usability: if the system forces the user to adopt an unacceptable mode of work then it is not usable.

There are, however, those who would dismiss our concentration on the task, saying that we do not even know enough about a theory of human tasks to support them in design. There is a good argument here (to which we return in Chapter 15). However, we can live with this confusion about what real tasks are because our understanding of tasks at the moment is sufficient to give us direction in design. The user's current tasks are studied and then supported by computers, which can in turn affect the nature of the original task and cause it to evolve. To illustrate, word processing has made it easy to manipulate paragraphs and reorder documents, allowing writers a completely new freedom that has affected writing styles. No longer is it vital to plan and construct text in an ordered fashion, since free-flowing prose can easily be restructured at a later date. This evolution of task in turn affects the design of the ideal system. However, we see this evolution as providing a motivating force behind the system development cycle, rather than a refutation of the whole idea of supportive design.

This word 'task' or the focus on accomplishing 'work' is also problematic when we think of areas such as domestic appliances, consumer electronics and e-commerce. There are three 'use' words that must all be true for a product to be successful; it must be:

useful – accomplish what is required: play music, cook dinner, format a document;

usable – do it easily and naturally, without danger of error, etc.;

used – make people want to use it, be attractive, engaging, fun, etc.

The last of these has not been a major factor until recently in HCI, but issues of motivation, enjoyment and experience are increasingly important. We are certainly even further from having a unified theory of experience than of task.

The question of whether HCI, or more importantly the design of interactive systems and the user interface in particular, is a science or a craft discipline is an interesting one. Does it involve artistic skill and fortuitous insight or reasoned methodical science? Here we can draw an analogy with architecture. The most impressive structures, the most beautiful buildings, the innovative and imaginative creations that provide aesthetic pleasure, all require inventive inspiration in design and a sense of artistry, and in this sense the discipline is a craft. However, these structures also have to be able to stand up to fulfill their purpose successfully, and to be able to do this the architect has to use science. So it is for HCI: beautiful and/or novel interfaces are artistically pleasing *and* capable of fulfilling the tasks required – a marriage of art and science into a successful whole. We want to reuse lessons learned from the past about how to achieve good results and avoid bad ones. For this we require both craft and science. Innovative ideas lead to more usable systems, but in order to maximize the potential benefit from the ideas, we need to understand not only that they work, but how and why they work. This scientific rationalization allows us to reuse related concepts in similar situations, in much the same way that architects can produce a bridge and know that it will stand, since it is based upon tried and tested principles.

The craft–science tension becomes even more difficult when we consider novel systems. Their increasing complexity means that our personal ideas of good and bad are no longer enough; for a complex system to be well designed we need to rely on something more than simply our intuition. Designers may be able to think about how one user would want to act, but how about groups? And what about new media? Our ideas of how best to share workloads or present video information are open to debate and question even in non-computing situations, and the incorporation of one version of good design into a computer system is quite likely to be unlike anyone else's version. Different people work in different ways, whilst different media color the nature of the interaction; both can dramatically change the very nature of the original task. In order to assist designers, it is unrealistic to assume that they can rely on artistic skill and perfect insight to develop usable systems. Instead we have to provide them with an understanding of the concepts involved, a scientific view of the reasons why certain things are successful whilst others are not, and then allow their creative nature to feed off this information: creative flow, underpinned with science; or maybe scientific method, accelerated by artistic insight. The truth is that HCI is required to be both a craft and a science in order to be successful.

HCI IN THE CURRICULUM

If HCI involves both craft and science then it must, in part at least, be taught. Imagination and skill may be qualities innate in the designer or developed through experience, but the underlying theory must be learned. In the past, when computers were used primarily by expert specialists, concentration on the interface was a luxury that was often relinquished. Now designers cannot afford to ignore the interface in favour of the functionality of their systems: the two are too closely intertwined. If the interface is poor, the functionality is obscured; if it is well designed, it will allow the system's functionality to support the user's task.

Increasingly, therefore, computer science educators cannot afford to ignore HCI. We would go as far as to claim that HCI should be integrated into every computer science or software engineering course, either as a recurring feature of other modules or, preferably, as a module itself. It should not be viewed as an 'optional extra' (although, of course, more advanced HCI options can complement a basic core course). This view is shared by the ACM SIGCHI curriculum development group, who propose a curriculum for such a core course [9]. The topics included in this book, although developed without reference to this curriculum, cover the main emphases of it, and include enough detail and coverage to support specialized options as well.

In courses other than computer science, HCI may well be an option specializing in a particular area, such as cognitive modeling or task analysis. Selected use of the relevant chapters of this book can also support such a course.

HCI must be taken seriously by designers and educators if the requirement for additional complexity in the system is to be matched by increased clarity and usability in the interface. In this book we demonstrate how this can be done in practice.

Design Focus

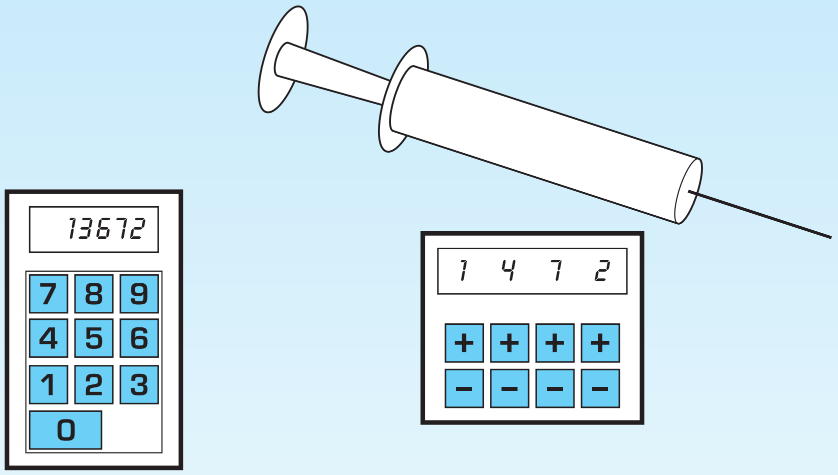
Quick fixes

You should expect to spend both time and money on interface design, just as you would with other parts of a system. So in one sense there are no quick fixes. However, a few simple steps can make a dramatic improvement.

Think 'user'
Probably 90% of the value of any interface design technique is that it forces the designer to remember that someone (and in particular someone else) will use the system under construction.

Try it out
Of course, many designers will build a system that they find easy and pleasant to use, and they find it incomprehensible that anyone else could have trouble with it. Simply sitting someone down with an early version of an interface (without the designer prompting them at each step!) is enormously valuable. Professional usability laboratories will have video equipment, one-way mirrors and other sophisticated monitors, but a notebook and pencil and a home-video camera will suffice (more about evaluation in Chapter 9).

Involve the users
Where possible, the eventual users should be involved in the design process. They have vital knowledge and will soon find flaws. A mechanical syringe was once being developed and a prototype was demonstrated to hospital staff. Happily they quickly noticed the potentially fatal flaw in its interface.



Before consulting users After consulting users

Figure 0.1 Automatic syringe: setting the dose to 1372. The effect of one key slip before and after user involvement

The doses were entered via a numeric keypad: an accidental keypress and the dose could be out by a factor of 10! The production version had individual increment/decrement buttons for each digit (more about participatory design in Chapter 13).

Iterate

People are complicated, so you won't get it right first time. Programming an interface can be a very difficult and time-consuming business. So, the result becomes precious and the builder will want to defend it and minimize changes. Making early prototypes less precious and easier to throw away is crucial. Happily there are now many interface builder tools that aid this process. For example, mock-ups can be quickly constructed using HyperCard on the Apple Macintosh or Visual Basic on the PC. For visual and layout decisions, paper designs and simple models can be used (more about iterative design in Chapter 5).

PART 1

Foundations

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

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

1 THE HUMAN

OVERVIEW

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

1.1 INTRODUCTION

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

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

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

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

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

1.2 INPUT–OUTPUT CHANNELS

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

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

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

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

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

1.2.1 Vision

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

The human eye

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

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

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

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

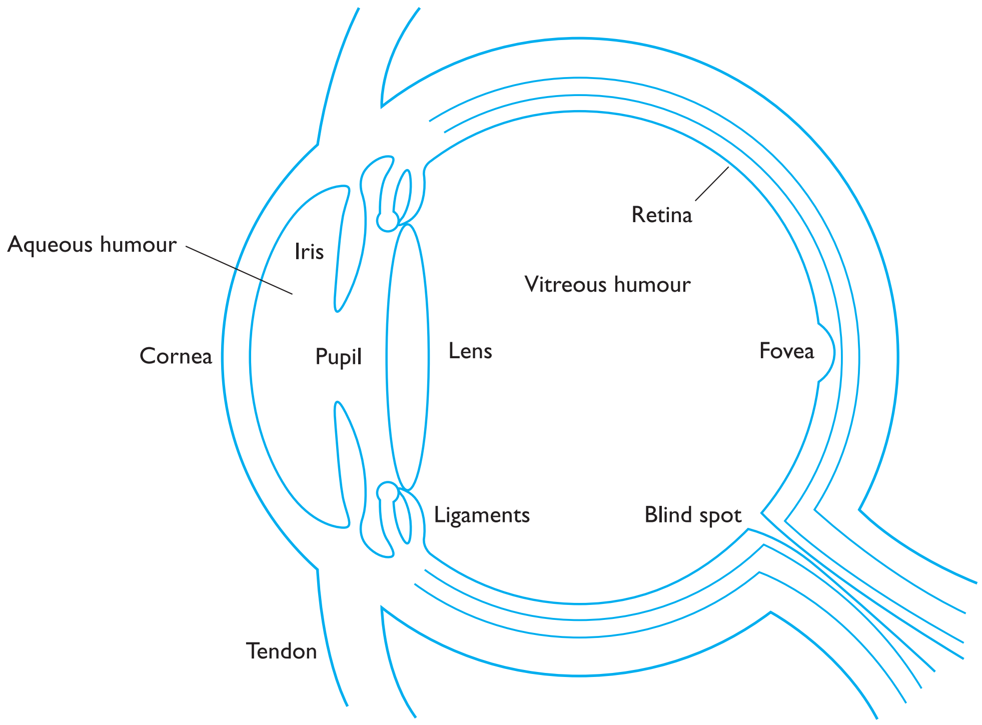


Figure 1.1 The human eye

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

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

Visual perception

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

Design Focus

Getting noticed

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

A B C D E F • H I J K

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

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

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

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

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

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

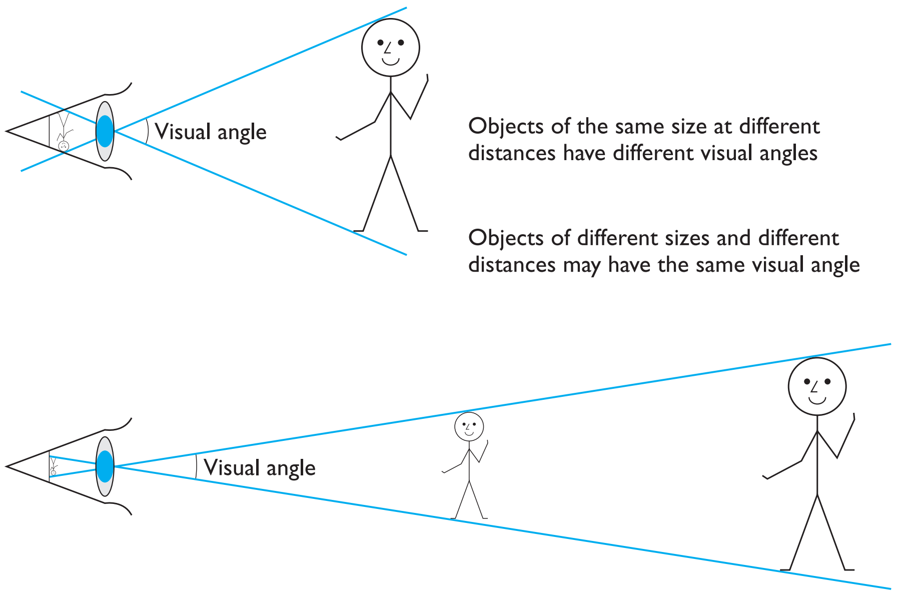

Visual angle
Objects of the same size at different distances have different visual angles
Objects of different sizes and different distances may have the same visual angle
Visual angle

Figure 1.2 Visual angle

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

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

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

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

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

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

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

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

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

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

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

The capabilities and limitations of visual processing

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

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

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

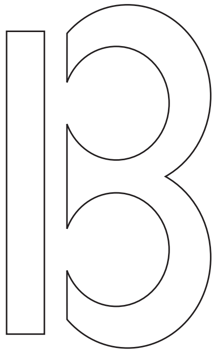


Figure 1.3 An ambiguous shape?

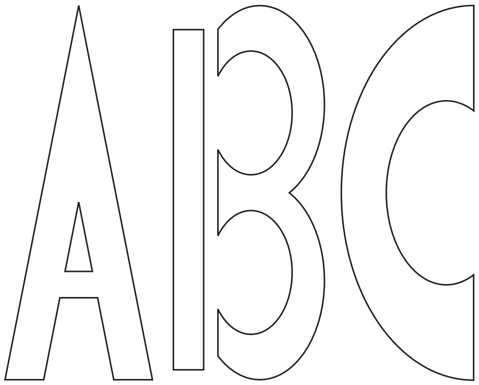


Figure 1.4 ABC

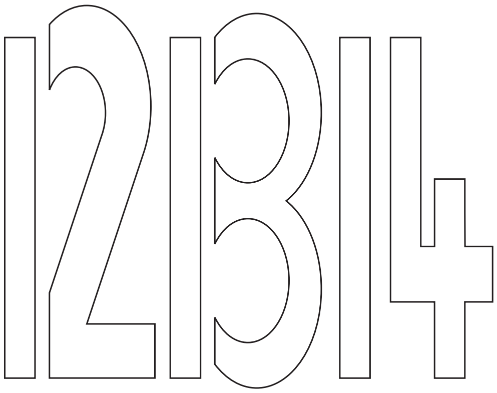


Figure 1.5 12 13 14

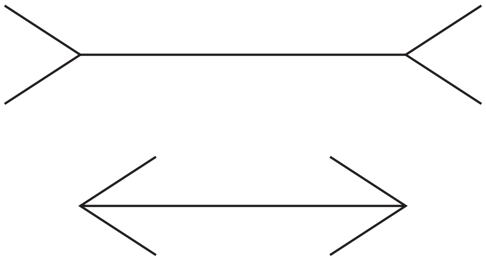


Figure 1.6 The Muller–Lyer illusion – which line is longer?

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

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

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

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

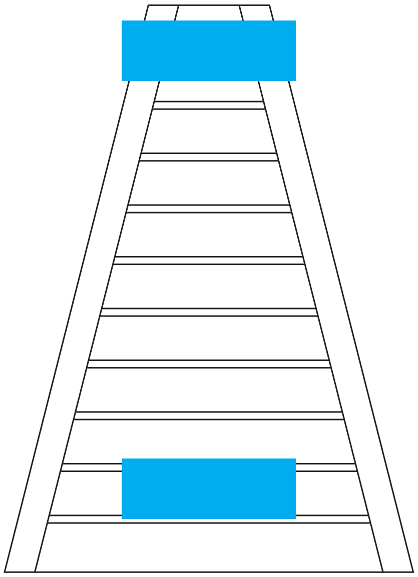


Figure 1.7 The Ponzo illusion – are these the same size?

The quick brown

fox jumps over the

the lazy dog.

Figure 1.8 Is this text correct?

Design Focus

Where's the middle?

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

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

Reading

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

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

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

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

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

1.2.2 Hearing

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

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

The human ear

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

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

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

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

Processing sound

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

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

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

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

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

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

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

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

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

1.2.3 Touch

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

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

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

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

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

Handling the goods

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

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

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

1.2.4 Movement

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

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

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

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

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

where a and b are empirically determined constants.

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

1.3 HUMAN MEMORY

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

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

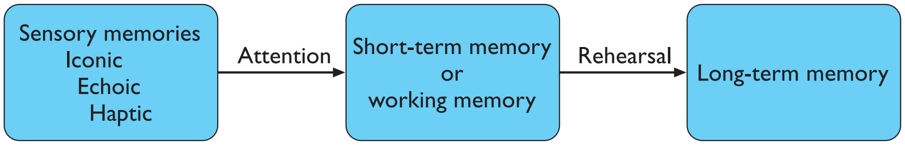


Figure 1.9 A model of the structure of memory

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

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

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

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

1.3.1 Sensory memory

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

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

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

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

1.3.2 Short-term memory

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

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

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

265397620853

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

Now try the following sequence:

44 113 245 8920

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

Design Focus

Cashing in

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

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

HEC ATR ANU PTH ETR EET

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

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

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

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

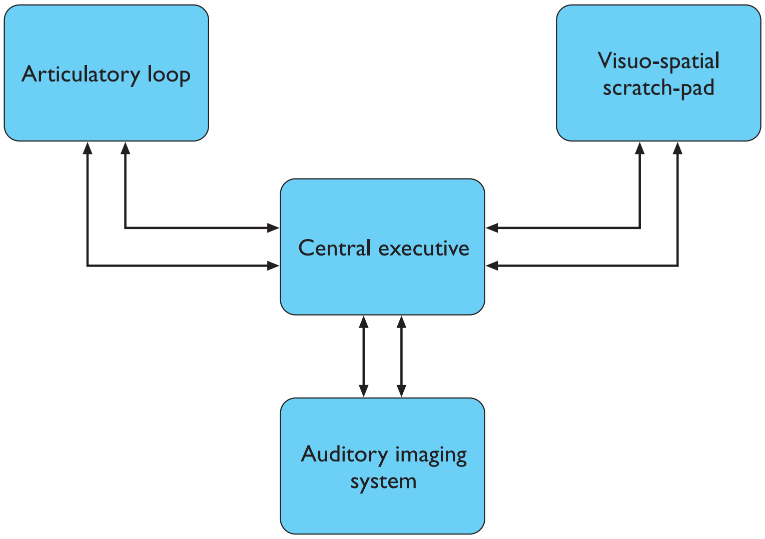


Figure 1.10 A more detailed model of short-term memory

DESIGN FOCUS

7 ± 2 revisited

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

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

1.3.3 Long-term memory

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

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

Long-term memory structure

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

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

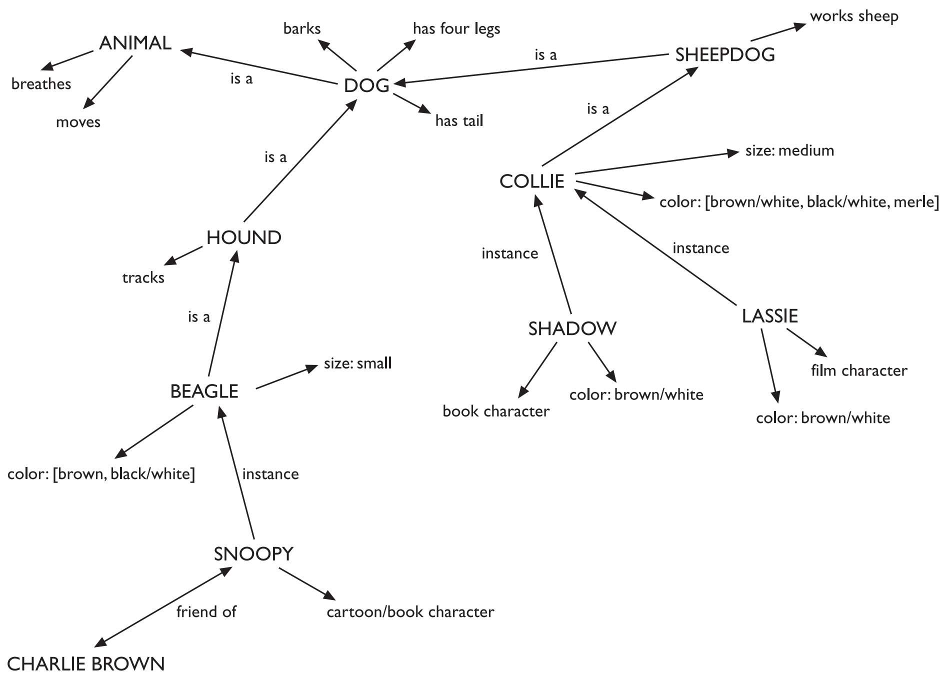


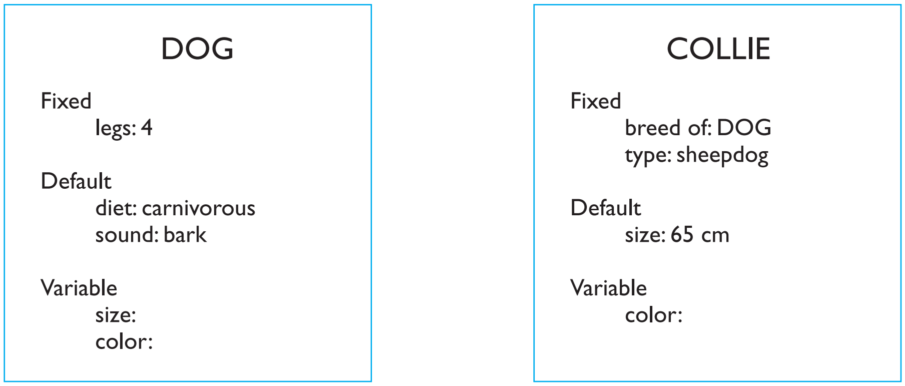
Figure 1.11 Long-term memory may store information in a semantic network

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

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

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

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


DOG

Fixed
legs: 4

Default
diet: carnivorous
sound: bark

Variable
size:
color:

COLLIE

Fixed
breed of: DOG
type: sheepdog

Default
size: 65 cm

Variable
color:

Figure 1.12 A frame-based representation of knowledge

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

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

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

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

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

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

Script for a visit to the vet

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

Figure 1.13 A script for visiting the vet

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

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

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

Props Objects involved in the events described in the script.

Roles Actions performed by particular participants.

Scenes The sequences of events that occur.

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

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

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

IF dog is wagging tail
THEN pat dog

IF dog is growling
THEN run away

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

Long-term memory processes

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

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

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

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

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

Now try list B.

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

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

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

Memorable or secure?

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

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

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

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

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

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

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

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

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

child red plane dog friend blood cold tree big angry

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

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

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

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

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

Improve your memory

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

Look at the list below of numbers and associated words:

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

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

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

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

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

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

1.4 THINKING: REASONING AND PROBLEM SOLVING

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

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

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

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

1.4.1 Reasoning

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

Deductive reasoning

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

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

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

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

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

Deductive reasoning is therefore often misapplied. Given the premises

Some people are babies
Some babies cry

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

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

Inductive reasoning

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

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

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

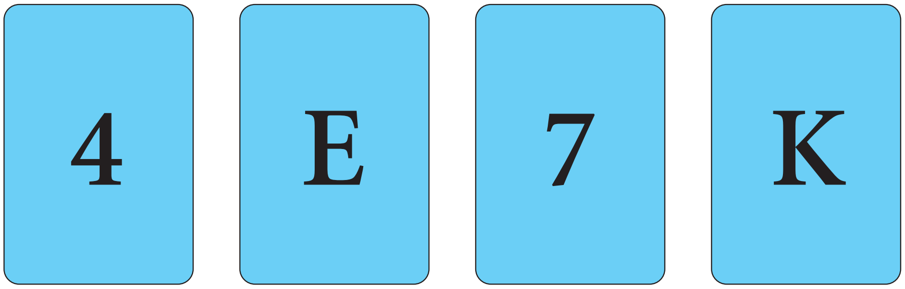


Figure 1.14 Wason's cards

Filling the gaps

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

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

Abductive reasoning

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

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

1.4.2 Problem solving

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

Gestalt theory

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

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

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

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

Problem space theory

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

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

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

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

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

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

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

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

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

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

Analogy in problem solving

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

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

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

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

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

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

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

1.4.3 Skill acquisition

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

Chess: of human and artificial intelligence

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

1.4.4 Errors and mental models

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

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

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

DESIGN FOCUS

Human error and false memories

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

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

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

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

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

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

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

Courtesy of popperfoto.com

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

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

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

1.5 EMOTION

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

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

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

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

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

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

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

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

1.6 INDIVIDUAL DIFFERENCES

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

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

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

1.7 PSYCHOLOGY AND THE DESIGN OF INTERACTIVE SYSTEMS

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

1.7.1 Guidelines

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

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

1.7.2 Models to support design

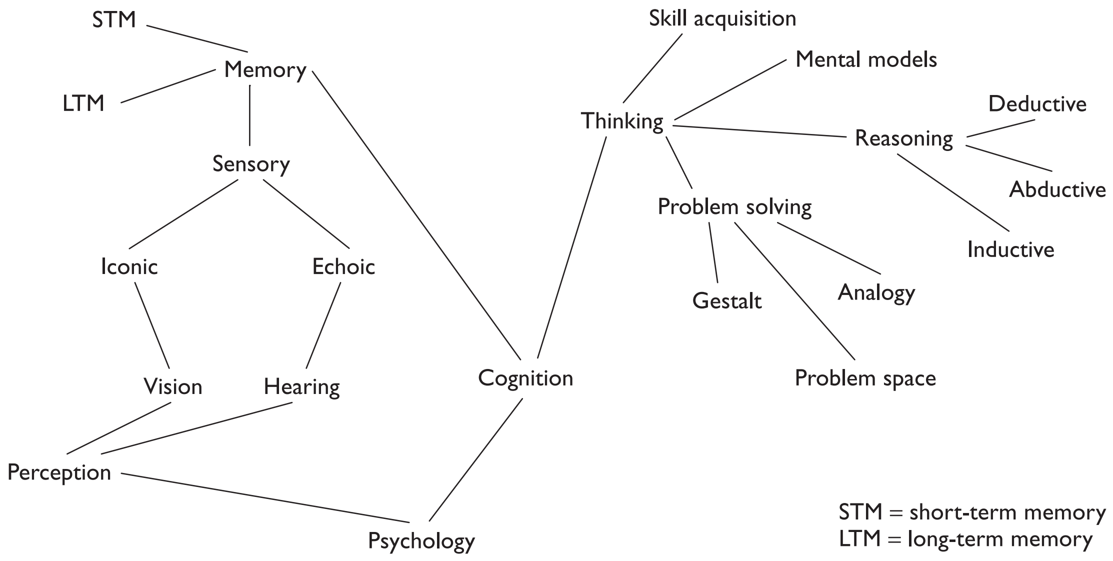
As well as guidelines and principles, psychological theory has led to the development of analytic and predictive models of user behavior. Some of these include a specific model of human problem solving, others of physical activity, and others attempt a more comprehensive view of cognition. Some predict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

1.7.3 Techniques for evaluation

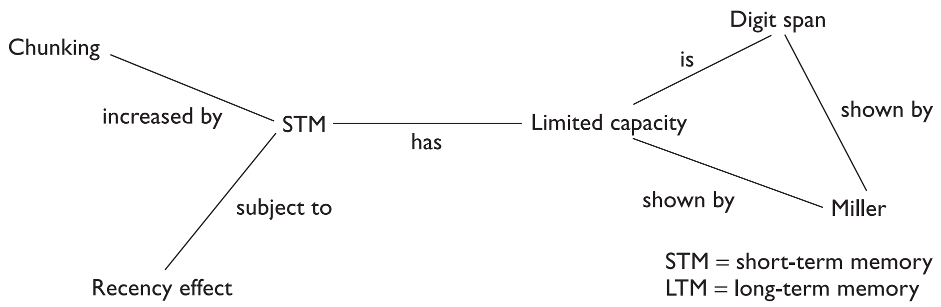
In addition to providing us with a wealth of theoretical understanding of the human user, psychology also provides a range of empirical techniques which we can employ to evaluate our designs and our systems. In order to use these effectively we need to understand the scope and benefits of each method.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

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

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



Top-down view



Bottom-up view

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

1.8 SUMMARY

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

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

EXERCISES

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

RECOMMENDED READING

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

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

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

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

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

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

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

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

2 THE COMPUTER

OVERVIEW

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

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

2.1 INTRODUCTION

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

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

Exercise: how many computers?

In a group or class do a quick survey:

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

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

Discuss your answers.

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

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

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

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

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

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

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

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

2.1.1 A typical computer system

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

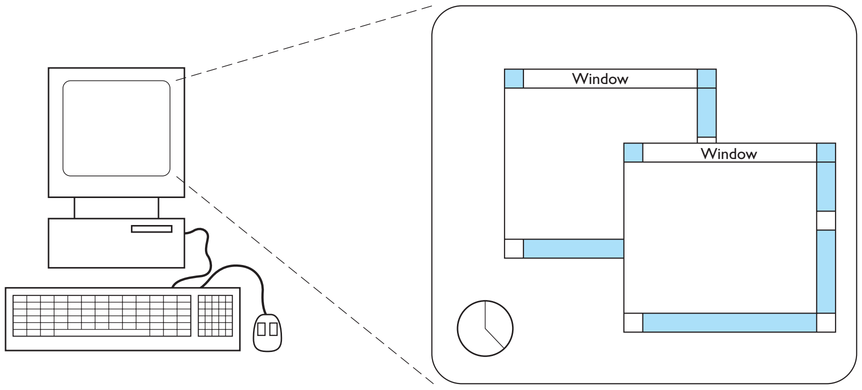


Figure 2.1 A typical computer system

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

2.1.2 Levels of interaction – batch processing

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

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

2.1.3 Richer interaction – everywhere, everywhen

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

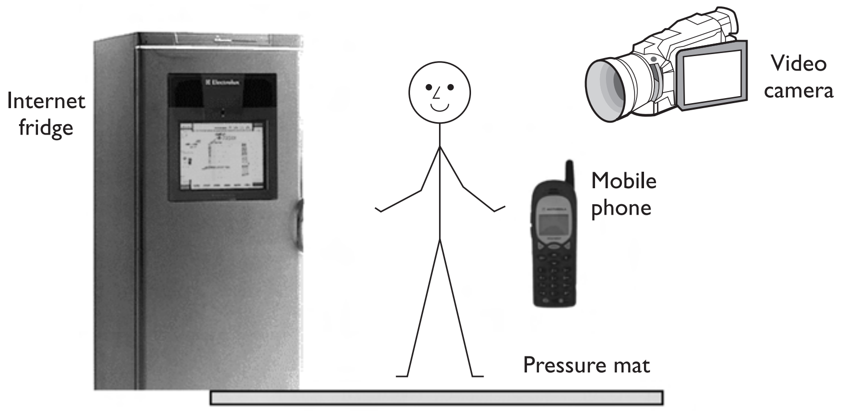


Figure 2.2 A typical computer system? Photo courtesy Electrolux

2.2 TEXT ENTRY DEVICES

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

2.2.1 The alphanumeric keyboard

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

The QWERTY keyboard

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

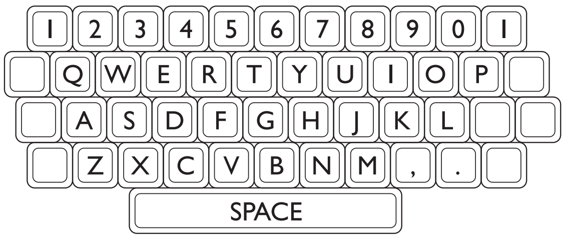


Figure 2.3 The standard QWERTY keyboard

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

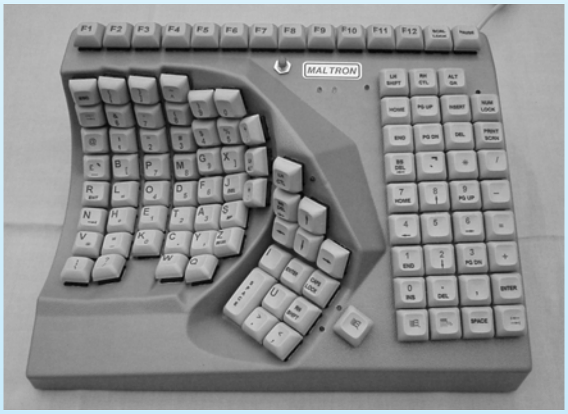
The electric typewriter and now the computer keyboard are not subject to the original mechanical constraints, but the QWERTY keyboard remains the dominant layout. The reason for this is social – the vast base of trained typists would be reluctant to relearn their craft, whilst the management is not prepared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

How keyboards work

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

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

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



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

Ease of learning – alphabetic keyboard

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

Ergonomics of use – DVORAK keyboard and split designs

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

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

2.2.2 Chord keyboards

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

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

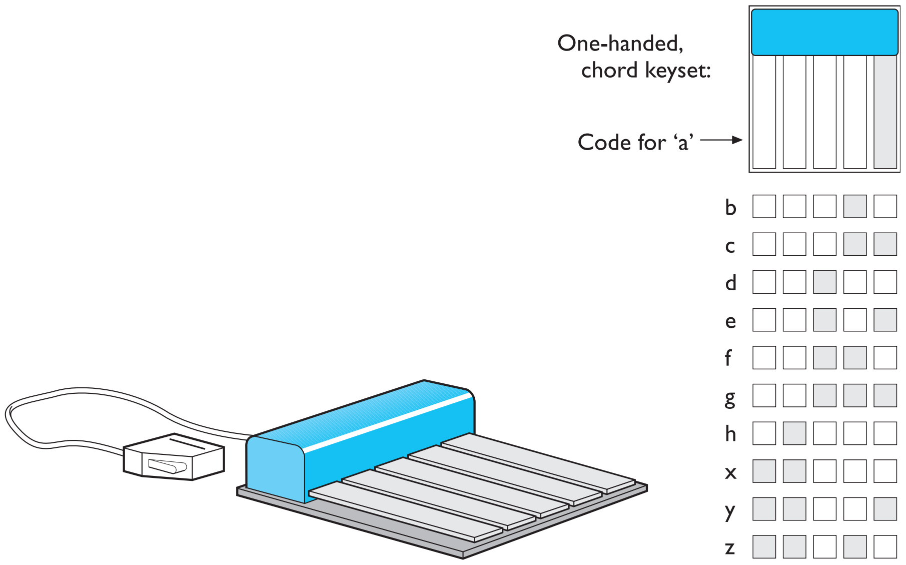


Figure 2.4 A very early chord keyboard (left) and its lettercodes (right)

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

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

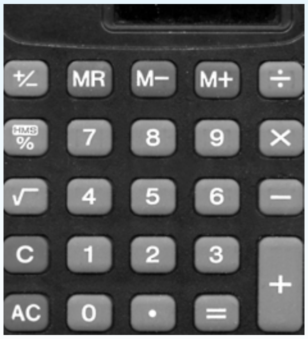
DESIGN FOCUS

Numeric keypads

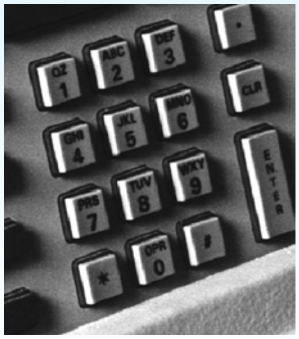
Alphanumeric keyboards (as the name suggests) include numbers as well as letters. In the QWERTY layout these are in a line across the top of the keyboard, but in most larger keyboards there is also a separate number pad to allow faster entry of digits. Number keypads occur in other contexts too, including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

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

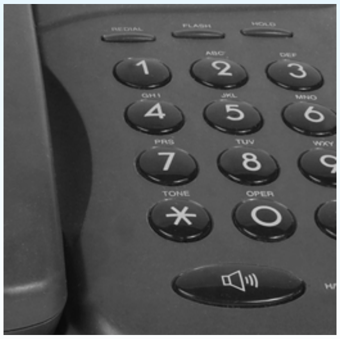
One of the authors was caught out by this once when he forgot the PIN number of his cash card. He half remembered the digits, but also his fingers knew where to type, so he 'practiced' on his calculator. Unfortunately ATMs use telephone-style layout!



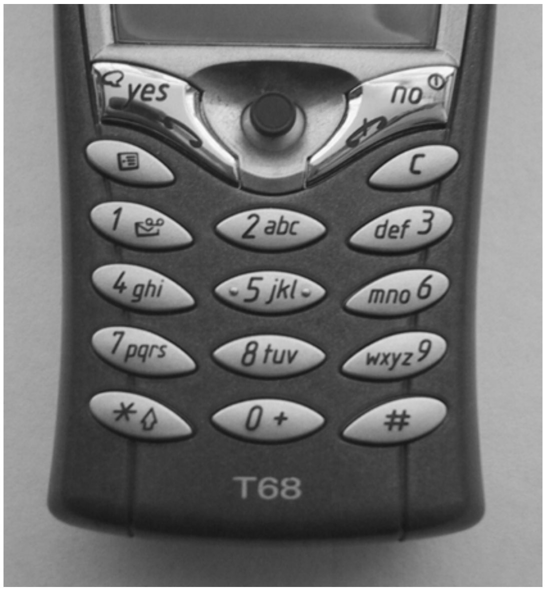
calculator



ATM



phone



Typical key mapping:

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

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

2.2.3 Phone pad and T9 entry

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

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

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

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

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

2.2.4 Handwriting recognition

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

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

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

Signature authentication

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

2.2.5 Speech recognition

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

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

2.3 POSITIONING, POINTING AND DRAWING

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

2.3.1 The mouse

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

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

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

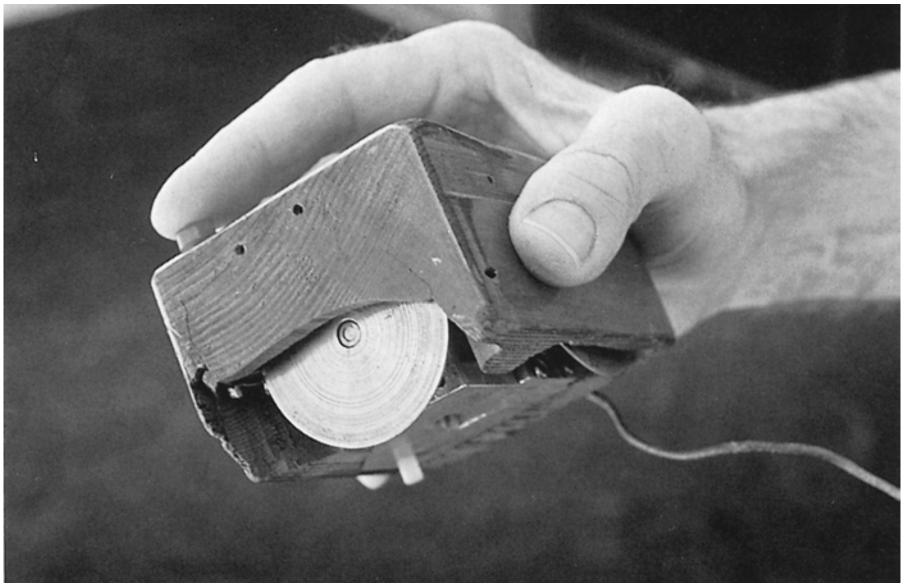


Figure 2.6 The first mouse. Photograph courtesy of Douglas Engelbart and Bootstrap Institute

Optical mice

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

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

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

2.3.2 Touchpad

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

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

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

2.3.3 Trackball and thumbwheel

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

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

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

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

2.3.4 Joystick and keyboard nipple

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

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

2.3.5 Touch-sensitive screens (touchscreens)

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

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

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

2.3.6 Stylus and light pen

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

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

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

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

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

2.3.7 Digitizing tablet

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

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

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

2.3.8 Eyegaze

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

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

2.3.9 Cursor keys and discrete positioning

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

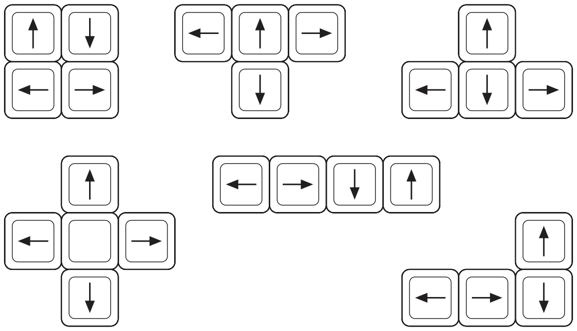


Figure 2.7 Various cursor key layouts

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

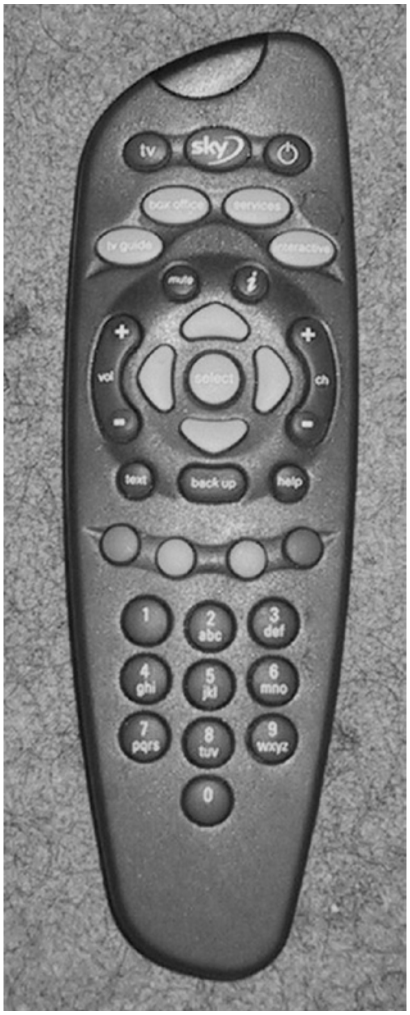
Cursor keys are available on most keyboards. Four keys on the keyboard are used to control the cursor, one each for up, down, left and right. There is no standardized layout for the keys. Some layouts are shown in Figure 2.7, but the most common now is the inverted 'T'.

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

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

2.4 DISPLAY DEVICES

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



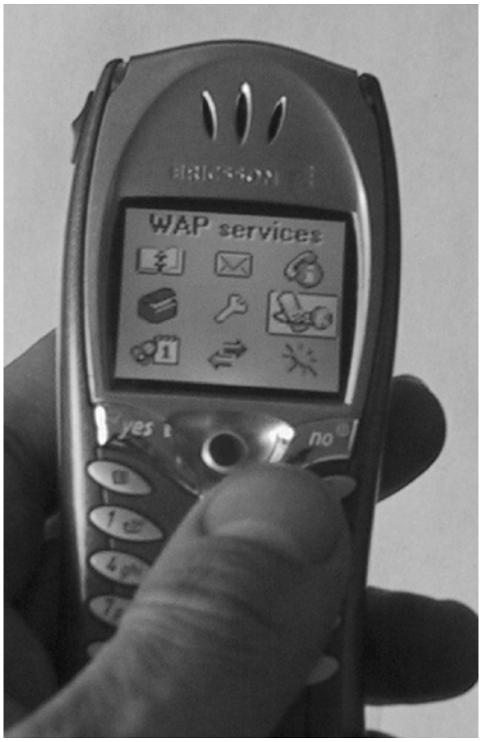


Figure 2.8 Satellite TV remote control and mobile phone. Source: Photograph left by Alan Dix with permission from British Sky Broadcasting Limited, photograph right by Alan Dix (Ericsson phone)

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

2.4.1 Bitmap displays – resolution and color

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

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

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

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

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

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

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

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

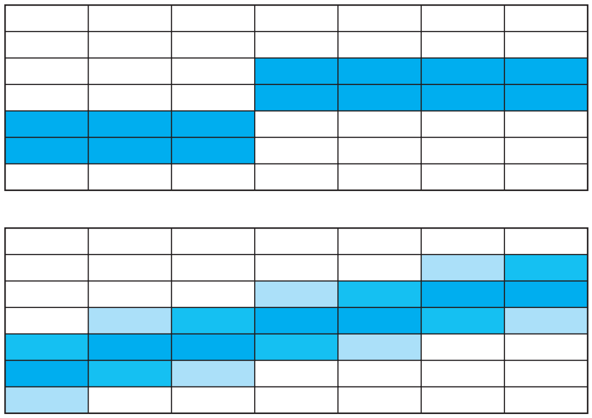


Figure 2.9 Magnified anti-aliased lines

2.4.2 Technologies

Cathode ray tube

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

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

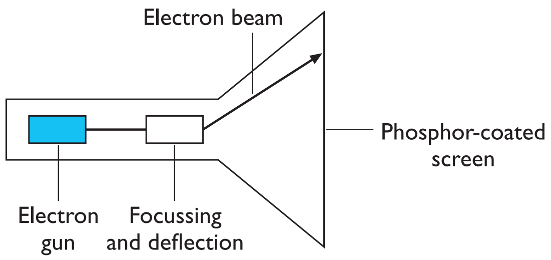


Figure 2.10 CRT screen

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

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

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

Health hazards of CRT displays

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

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

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

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

Liquid crystal display

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

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

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

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

Special displays

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

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

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

2.4.3 Large displays and situated displays

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

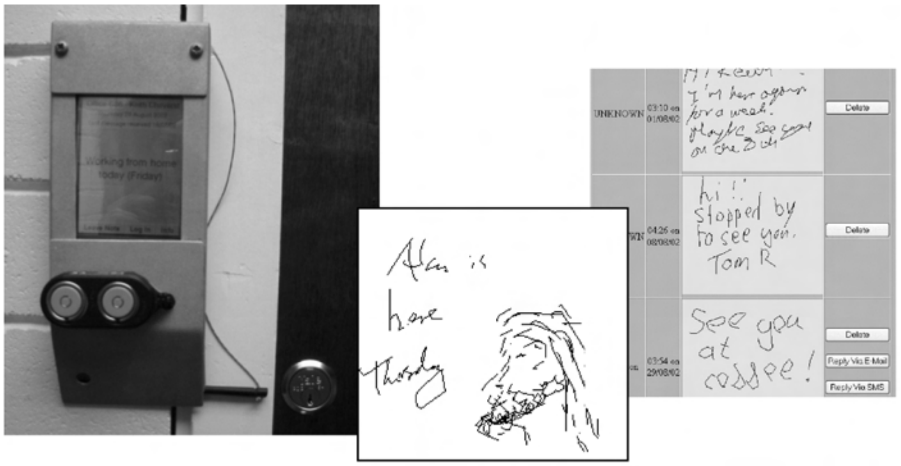
There are several types of large screen display. Some use gas plasma technology to create large flat bitmap displays. These behave just like a normal screen except they are big and usually have the HDTV (high definition television) wide screen format which has an aspect ratio of 16:9 instead of the 4:3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

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

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

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

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



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

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

Design Focus

Hermes: a situated display

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

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

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

2.4.4 Digital paper

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

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

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

2.5 DEVICES FOR VIRTUAL REALITY AND 3D INTERACTION

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

2.5.1 Positioning in 3D space

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

Cockpit and virtual controls

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

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

The 3D mouse

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

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

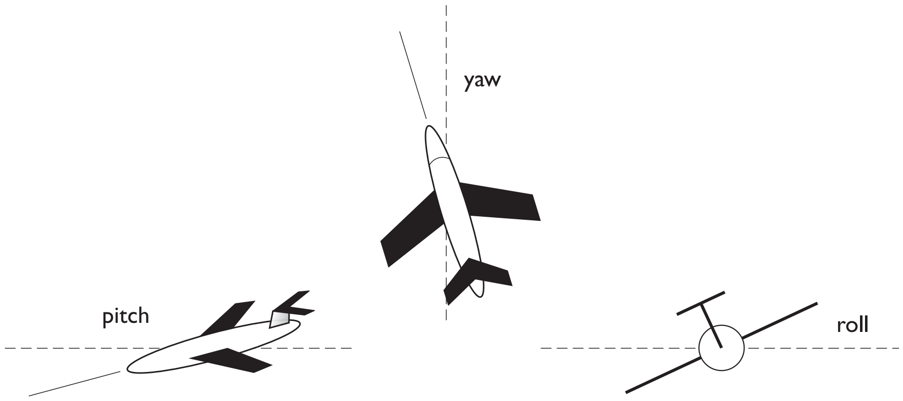


Figure 2.12 Pitch, yaw and roll

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

Dataglove

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

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

Virtual reality helmets

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

Whole-body tracking

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

2.5.2 3D displays

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

Seeing in 3D

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

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

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

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

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

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

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

VR motion sickness

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

Simulators and VR caves

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

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

2.6 PHYSICAL CONTROLS, SENSORS AND SPECIAL DEVICES

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

2.6.1 Special displays

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

2.6.2 Sound output

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

2.6.3 Touch, feel and smell

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

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

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

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

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

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

DESIGN FOCUS

Feeling the road

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

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

2.6.4 Physical controls

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

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

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

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

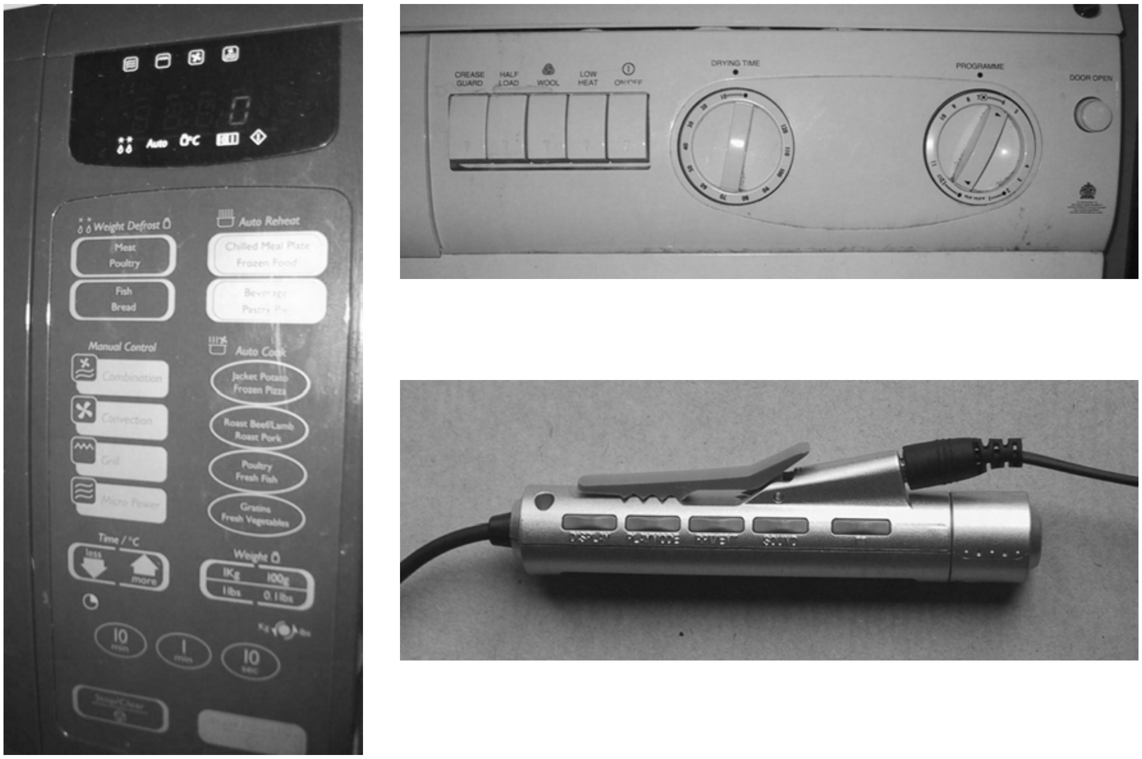


Figure 2.13 Physical controls on microwave, washing machine and MiniDisc. Source: Photograph bottom right by Alan Dix with permission from Sony (UK)

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

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

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

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

2.6.5 Environment and bio-sensing

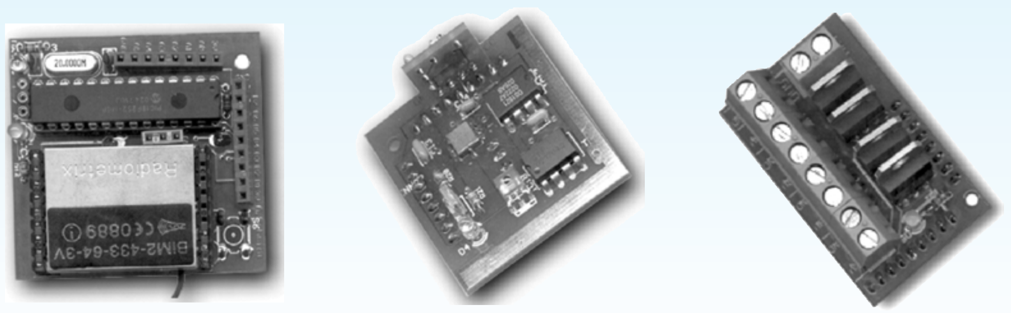
In a public washroom there are often no controls for the wash basins, you simply put your hands underneath and (hope that) the water flows. Similarly when you open the door of a car, the courtesy light turns on. The washbasin is controlled by a small infrared sensor that is triggered when your hands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

DESIGN FOCUS

Smart-Its – making using sensors easy

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

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



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

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

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

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

2.7 PAPER: PRINTING AND SCANNING

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

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

2.7.1 Printing

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

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

Common types of dot-based printers

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

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

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

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

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

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

Printing in the workplace

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

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

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

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

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

2.7.2 Fonts and page description languages

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

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

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

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

Courier is a fixed-pitch font

Times Roman is a variable-pitch serif font

Minion is also a variable-pitch serif font

Gill Sans is a variable-pitch sans-serif font

A mathematics font: αβξ±π∈∀∞⊥≠ℵ∂√∃

Figure 2.14 Examples of different fonts

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

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

Design Focus

Readability of text

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

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

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

2.7.3 Screen and page

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

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

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

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

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

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

2.7.4 Scanners and optical character recognition

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

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

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

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

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

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

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

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

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

Paper-based interaction

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

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

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

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

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

Worked exercise

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

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

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

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

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

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

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

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

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

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

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

2.8 MEMORY

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

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

2.8.1 RAM and short-term memory (STM)

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

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

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

2.8.2 Disks and long-term memory (LTM)

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

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

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

2.8.3 Understanding speed and capacity

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

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

Table 2.1 Typical capacities of different storage media

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

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

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

Technological change and storage capacity

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

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

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

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

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

2.8.4 Compression

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

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

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

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

2.8.5 Storage format and standards

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

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

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

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

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

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

2.8.6 Methods of access

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

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

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

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

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

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

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

Example

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

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

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

2.9 PROCESSING AND NETWORKS

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

Moore's law

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

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

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

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

2.9.1 Effects of finite processor speed

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

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

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

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

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

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

DESIGN FOCUS

The myth of the infinitely fast machine

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

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

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

2.9.2 Limitations on interactive performance

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

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

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

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

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

2.9.3 Networked computing

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

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

The internet

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

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

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

Worked exercise

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

Answer

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

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

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

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

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

2.10 SUMMARY

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

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

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

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

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

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

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

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

EXERCISES

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

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

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

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

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

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

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

2.4 What is the myth of the infinitely fast machine?

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

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

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

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

RECOMMENDED READING

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

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

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

3 The interaction

OVERVIEW

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

3.1 INTRODUCTION

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

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

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

3.2 MODELS OF INTERACTION

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

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

3.2.1 The terms of interaction

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

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

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

3.2.2 The execution–evaluation cycle

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

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

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

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

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

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

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

Human error – slips and mistakes

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

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

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

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

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

3.2.3 The interaction framework

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

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

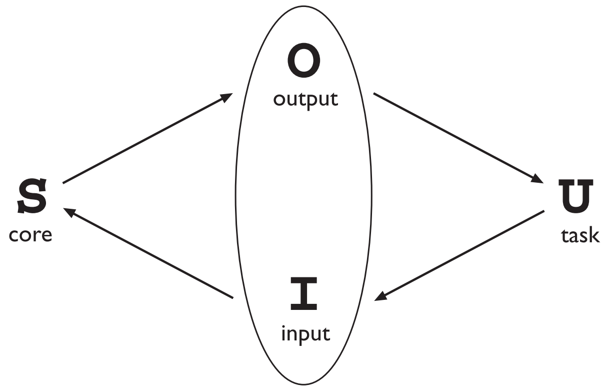


Figure 3.1 The general interaction framework

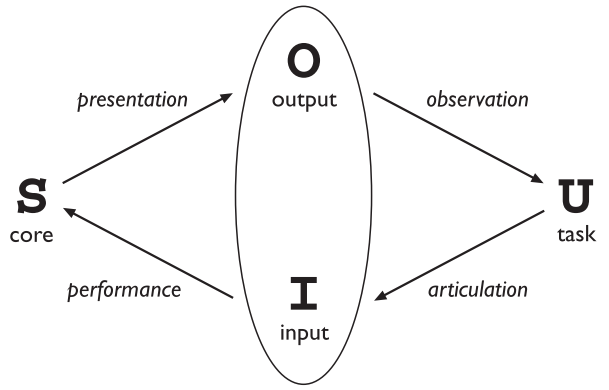


Figure 3.2 Translations between components

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

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

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

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

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

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

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

Assessing overall interaction

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

DESIGN FOCUS

Video recorder

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

3.3 FRAMEWORKS AND HCI

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

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

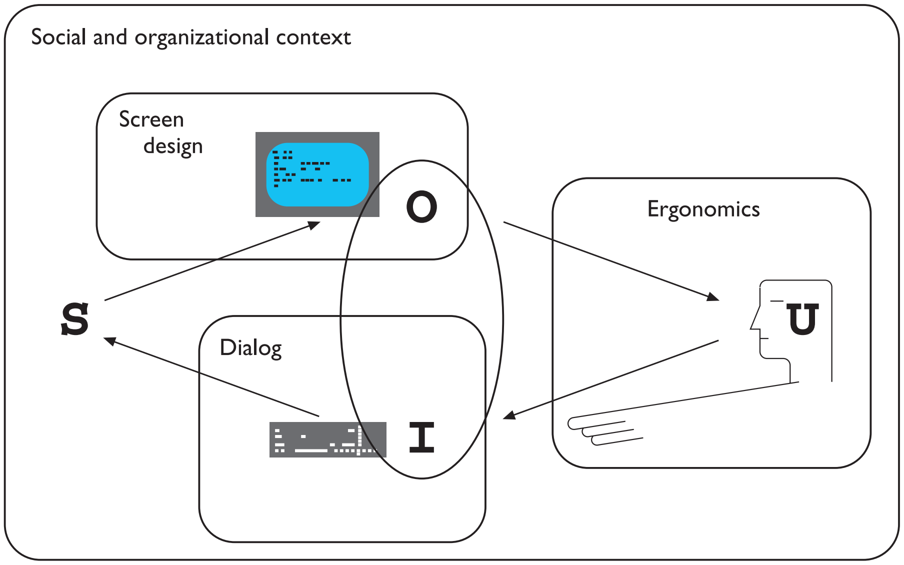


Figure 3.3 A framework for human–computer interaction. Adapted from ACM SIGCHI Curriculum Development Group [9]

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

3.4 ERGONOMICS

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

3.4.1 Arrangement of controls and displays

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

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

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

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

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

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

DESIGN FOCUS

Industrial interfaces

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

Glass interfaces vs. dials and knobs

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

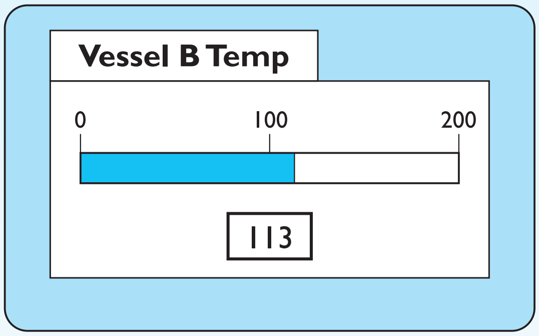


Figure 3.4 Multiple representations of the same information

Indirect manipulation

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

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

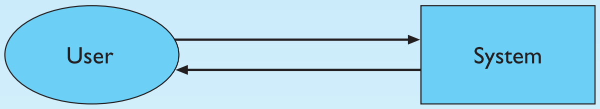


Figure 3.5 Office system – direct manipulation

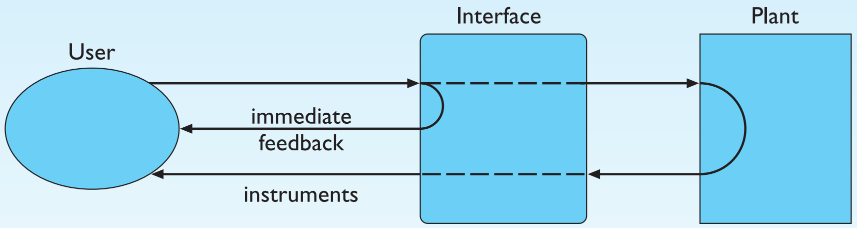


Figure 3.6 Indirect manipulation – two kinds of feedback

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

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

3.4.2 The physical environment of the interaction

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

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

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

3.4.3 Health issues

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

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

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

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

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

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

3.4.4 The use of color

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

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

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

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

3.4.5 Ergonomics and HCI

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

3.5 INTERACTION STYLES

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

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

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

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

Figure 3.7 Command line interface

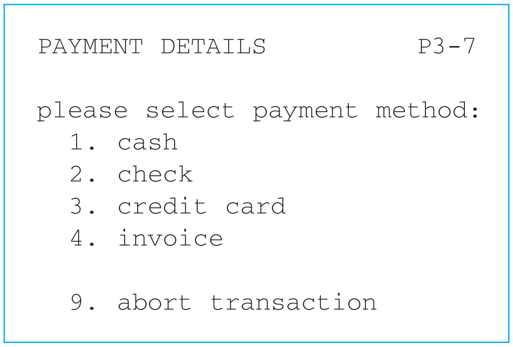
3.5.1 Command line interface

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

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

3.5.2 Menus

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



```
PAYMENT DETAILS              P3-7

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

   9. abort transaction
```

Figure 3.8 Menu-driven interface

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

3.5.3 Natural language

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

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

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

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

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

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

3.5.4 Question/answer and query dialog

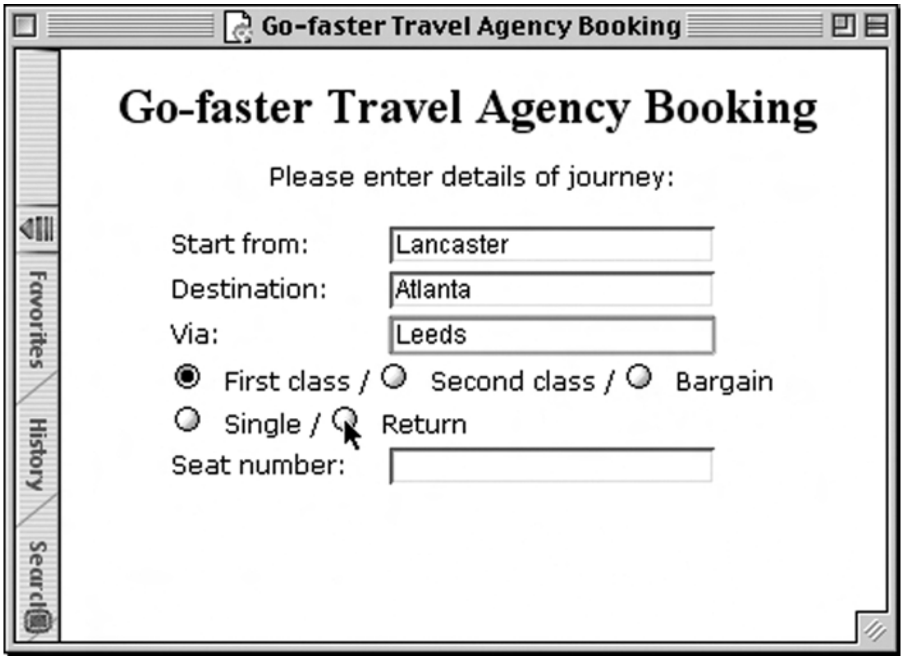
Question and answer dialog is a simple mechanism for providing input to an application in a specific domain. The user is asked a series of questions (mainly with yes/no responses, multiple choice, or codes) and so is led through the interaction step by step. An example of this would be web questionnaires.

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

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

3.5.5 Form-fills and spreadsheets

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



Go-faster Travel Agency Booking

Please enter details of journey:

Start from:	Lancaster
Destination:	Atlanta
Via:	Leeds

[✓ First class] / Second class / Bargain

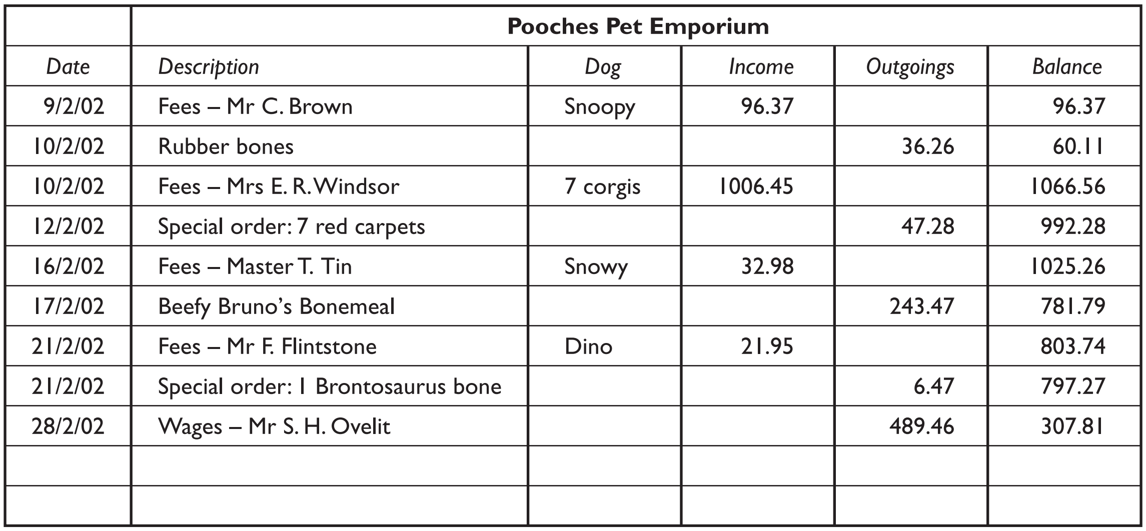
Single / Return

Seat number:

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

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

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



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

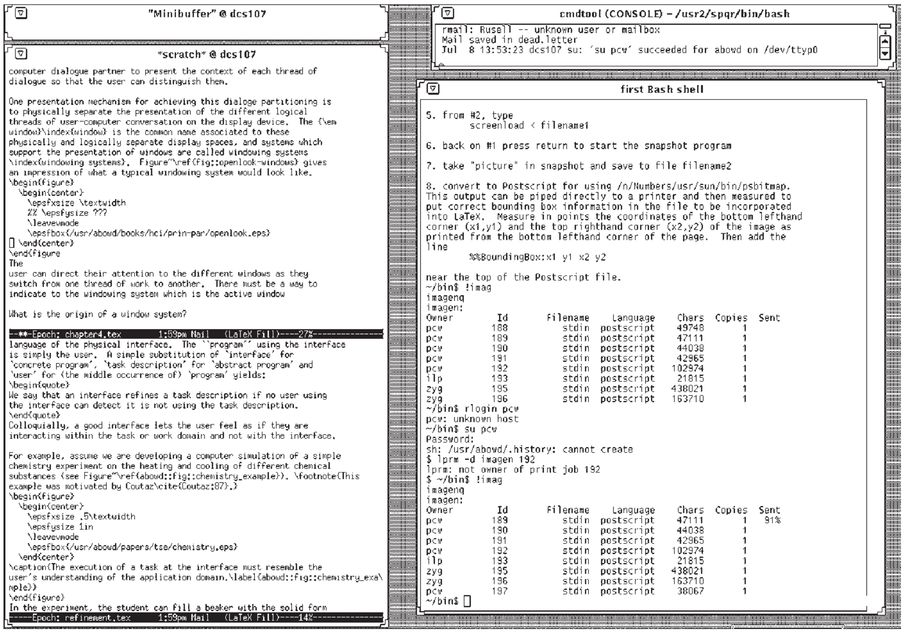
Figure 3.10 A typical spreadsheet

3.5.6 The WIMP interface

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

Mixing styles

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



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

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

What is the origin of a window system?

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

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

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

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

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

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

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

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

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

3.5.7 Point-and-click interfaces

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

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

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

3.5.8 Three-dimensional interfaces

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

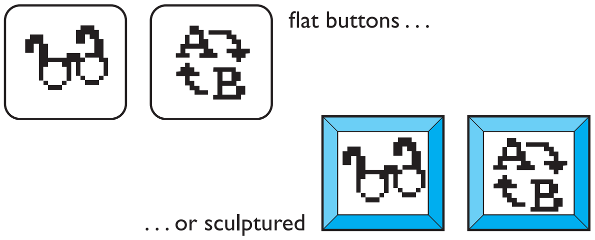


Figure 3.12 Buttons in 3D say 'press me'

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

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

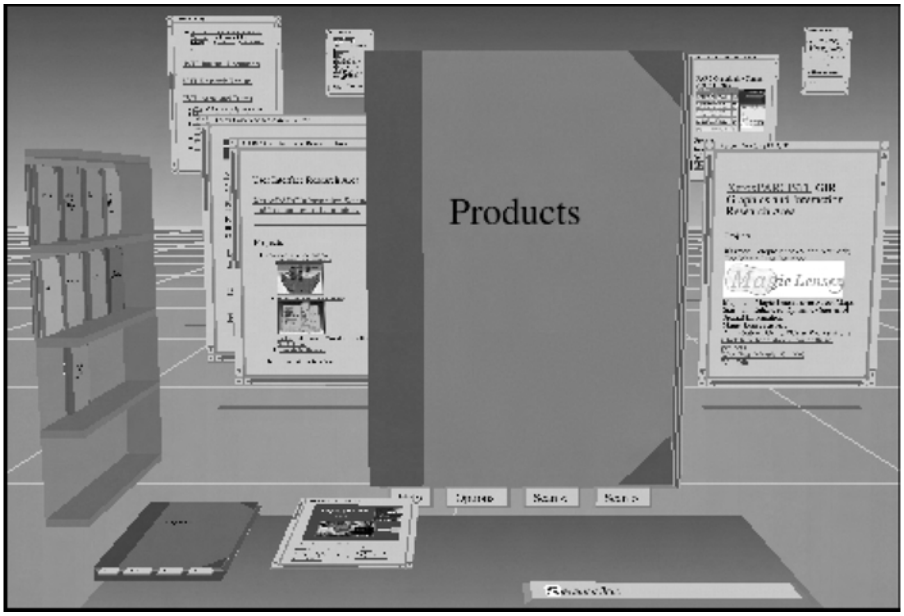


Figure 3.13 WebBook – using 3D to make more space (Card S.K., Robertson G.G. and York W. (1996). The WebBook and the Web Forager: An Information workspace for the World-Wide Web. *CHI96 Conference Proceedings*, 111–17. Copyright © 1996 ACM, Inc. Reprinted by permission)

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

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

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

Design Focus

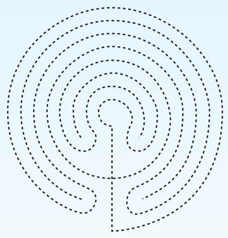
Navigation in 3D and 2D

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

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

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

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



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

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

3.6 ELEMENTS OF THE WIMP INTERFACE

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

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

3.6.1 Windows

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

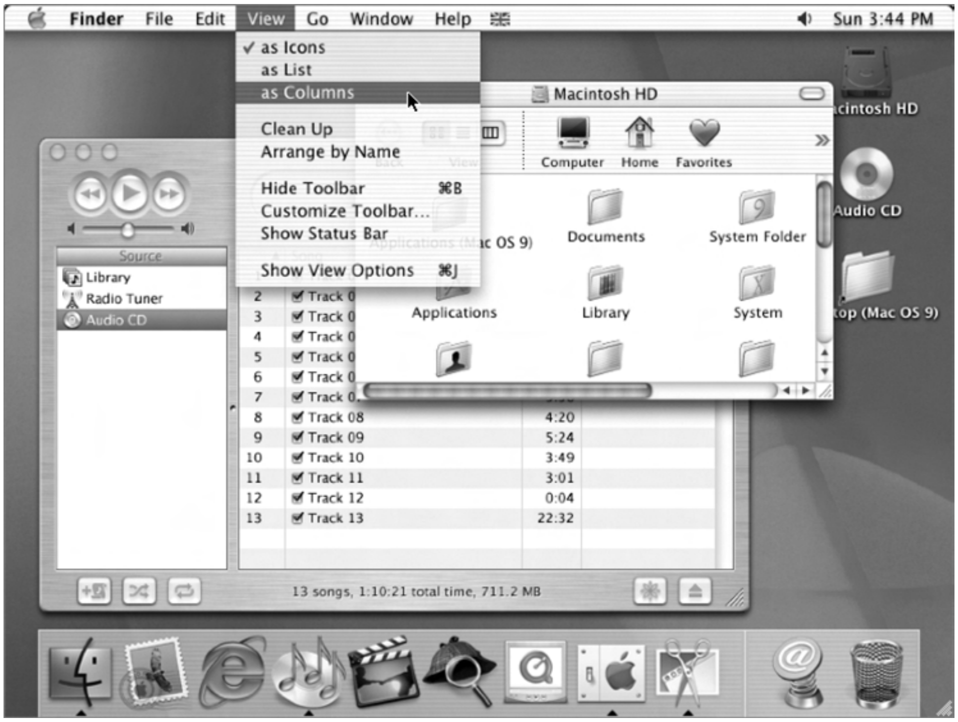


Figure 3.14 Elements of the WIMP interface – Microsoft Word 5.1 on an Apple Macintosh. Screen shot reprinted by permission from Apple Computer, Inc.

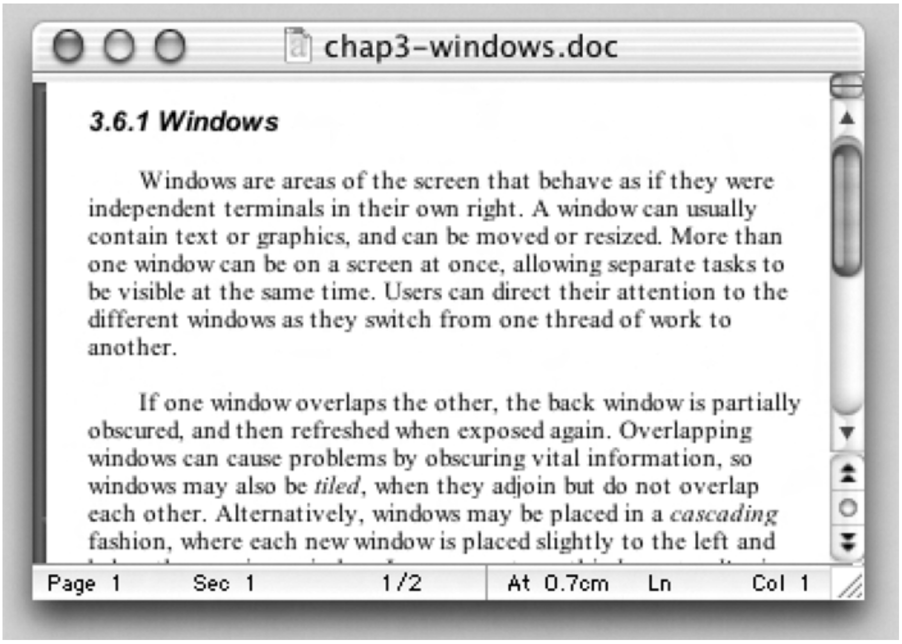
or resized. More than one window can be on a screen at once, allowing separate tasks to be visible at the same time. Users can direct their attention to the different windows as they switch from one thread of work to another.

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

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

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

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



3.6.1 Windows

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

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

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

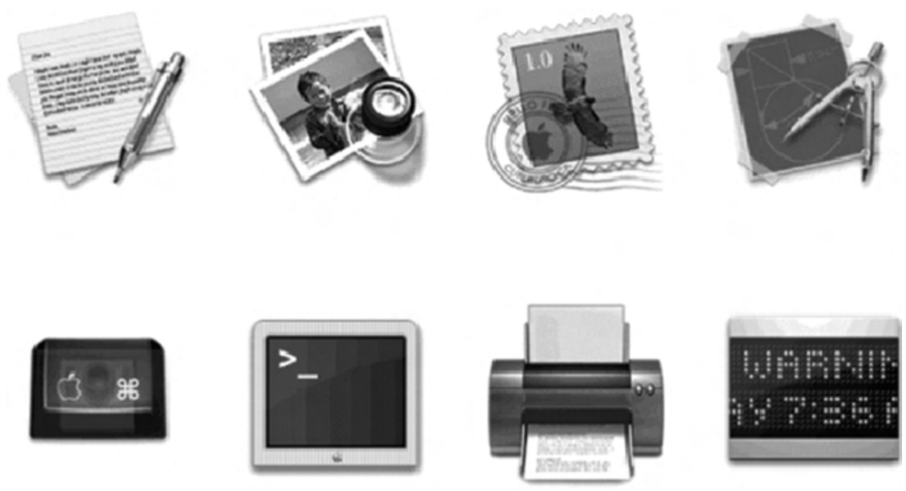


Figure 3.16 A variety of icons. Screen shot reprinted by permission from Apple Computer, Inc.

3.6.2 Icons

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

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

3.6.3 Pointers

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

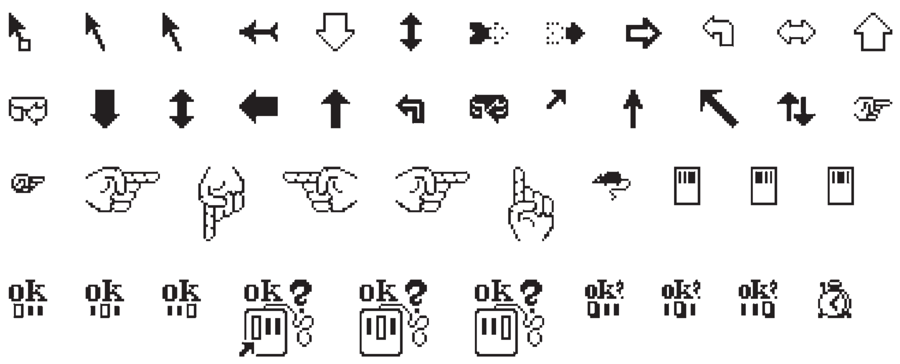


Figure 3.17 A variety of pointer cursors. Source: Sun Microsystems, Inc.

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

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

3.6.4 Menus

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

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

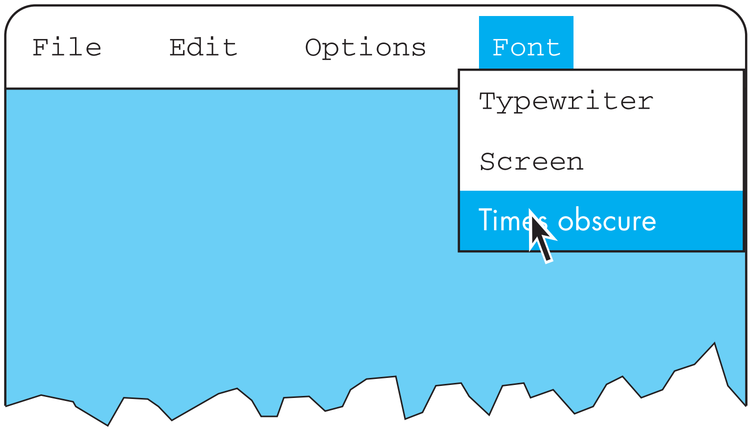


Figure 3.18 Pull-down menu

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

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

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

Keyboard accelerators

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

```
Find          F3
```

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

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

3.6.5 Buttons

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

3.6.6 Toolbars

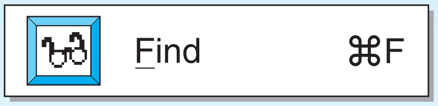
Many systems have a collection of small buttons, each with icons, placed at the top or side of the window and offering commonly used functions. The function of this *toolbar* is similar to a menu bar, but as the icons are smaller than the equivalent text more functions can be simultaneously displayed.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

Design Focus

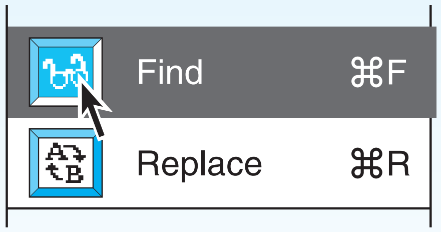
Learning toolbars

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

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



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



Selecting the menu option = selecting the icon

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

3.6.7 Palettes

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

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

3.6.8 Dialog boxes

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

3.7 INTERACTIVITY

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

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

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

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

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

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

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

3.8 THE CONTEXT OF THE INTERACTION

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

Bank managers don't type . . .

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

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

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

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

Design Focus

Half the picture?

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

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

3.9 EXPERIENCE, ENGAGEMENT AND FUN

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

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

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

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

3.9.1 Understanding experience

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

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

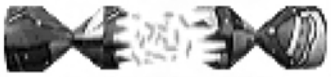
Csikszentimihalyi [82] looked at extreme experiences such as climbing a rock face in order to understand that feeling of total engagement that can sometimes happen. He calls this *flow* and it is perhaps related to what some sportspeople refer to as being ‘in the zone’. This sense of flow occurs when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

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

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

3.9.2 Designing experience

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



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

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

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

Table 3.1 The crackers experience [101]

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

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

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

3.9.3 Physical design and engagement

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

Designers are faced with many constraints:

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

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

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

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

Aesthetic The controls must look good.

Economic It must not cost too much!

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

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

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

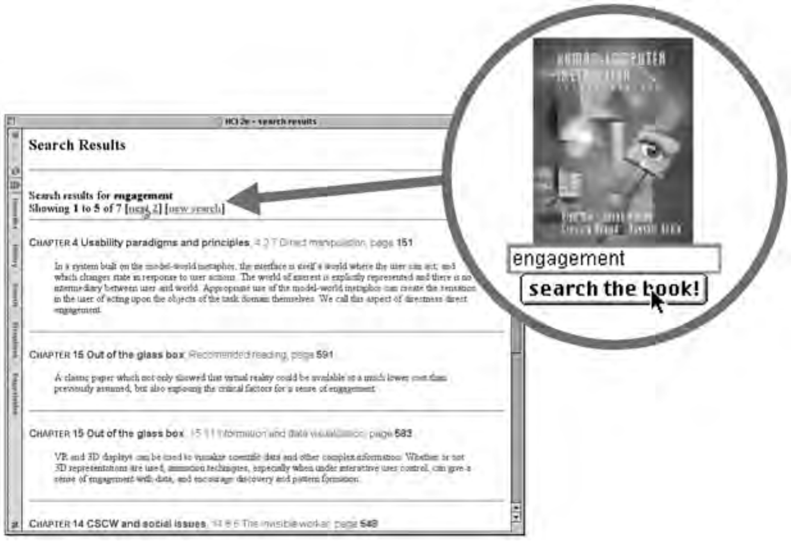
This is related closely to the idea of *affordances*, which we discuss in Section 5.7.2. The knob at the end of the MiniDisc controller affords turning – it is an obvious thing to do. However, this may not have mapped naturally onto the logical functions. Two of the press buttons are for cycling round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

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

3.9.4 Managing value

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

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



HCI 2e - search results

Search Results

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

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

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

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

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

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

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

CHAPTER 14 CSCW and social issues 14.6.6 The invisible worker, page 548

Figure 3.19 The web-based book search facility. Screen shot frame reprinted by permission from Microsoft Corporation

gains sooner as well as later, or at least produce a very good demonstration of potential future gains so that they have a *perceived* current value.

When we were preparing the website for the second edition of this book we thought very hard about how to give things that were of value to those who had the book, and also to those who hadn't. The latter is partly because we are all academics and researchers in the field and so want to contribute to the HCI community, but also of course we would like lots of people to buy the book. One option we thought of was to put the text online, which would be good for people without the book, but this would have less value to people who have the book (they might even be annoyed that those who hadn't paid should have access). The search mechanism was the result of this process (Figure 3.19). It gives value to those who have the book because it is a way of finding things. It is of value to those who don't because it acts as a sort of online encyclopedia of HCI. However, because it always gives the chapter and page number in the book it also says to those who haven't got the book: 'buy me'. See an extended case study about the design of the book search on the website at /e3/casestudy/search/

3.10 SUMMARY

In this chapter, we have looked at the interaction between human and computer, and, in particular, how we can ensure that the interaction is effective to allow the user to get the required job done. We have seen how we can use Norman's execution–evaluation model, and the interaction framework that extends it, to analyze the interaction in terms of how easy or difficult it is for the user to express what he wants and determine whether it has been done.

We have also looked at the role of ergonomics in interface design, in analyzing the physical characteristics of the interaction, and we have discussed a number of interface styles. We have considered how each of these factors can influence the effectiveness of the interaction.

Interactivity is at the heart of all modern interfaces and is important at many levels. Interaction between user and computer does not take place in a vacuum, but is affected by numerous social and organizational factors. These may be beyond the designer's control, but awareness of them can help to limit any negative effects on the interaction.

EXERCISES

3.1 Choose two of the interface styles (described in Section 3.5) that you have experience of using. Use the interaction framework to analyze the interaction involved in using these interface styles for a database selection task. Which of the distances is greatest in each case?

3.2 Find out all you can about natural language interfaces. Are there any successful systems? For what applications are these most appropriate?

3.3 What influence does the social environment in which you work have on your interaction with the computer? What effect does the organization (commercial or academic) to which you belong have on the interaction?

3.4 (a) Group the following functions under appropriate headings, assuming that they are to form the basis for a menu-driven word-processing system – the headings you choose will become the menu titles, with the functions appearing under the appropriate one. You can choose as many or as few menu headings as you wish. You may also alter the wordings of the functions slightly if you wish.

save, save as, new, delete, open mail, send mail, quit, undo, table, glossary, preferences, character style, format paragraph, lay out document, position on page, plain text, bold text, italic text, underline, open file, close file, open copy of file, increase point size, decrease point size, change font, add footnote, cut, copy, paste, clear, repaginate, add page break, insert graphic, insert index entry, print, print preview, page setup, view page, find word, change word, go to, go back, check spelling, view index, see table of contents, count words, renumber pages, repeat edit, show alternative document, help

(b) If possible, show someone else your headings, and ask them to group the functions under your headings. Compare their groupings with yours. You should find that there are areas of great similarity, and some differences. Discuss the similarities and discrepancies.

Why do some functions always seem to be grouped together?
Why do some groups of functions always get categorized correctly?
Why are some less easy to place under the 'correct' heading?
Why is this important?

3.5 Using your function groupings from Exercise 3.4, count the number of items in your menus.

(a) What is the average?
What is the disadvantage of putting all the functions on the screen at once?
What is the problem with using lots of menu headings?
What is the problem of using very few menu headings?

Consider the following: I can group my functions either into three menus, with lots of functions in each one, or into eight menus with fewer in each. Which will be easier to use? Why?

(b) *Optional experiment*
Design an experiment to test your answers. Perform the experiment and report on your results.

3.6 Describe (in words as well as graphically) the interaction framework introduced in *Human–Computer Interaction*. Show how it can be used to explain problems in the dialog between a user and a computer.

3.7 Describe briefly four different interaction styles used to accommodate the dialog between user and computer.

3.8 The typical computer screen has a WIMP setup (what does WIMP stand for?). Most common WIMP arrangements work on the basis of a desktop metaphor, in which common actions are likened to similar actions in the real world. For example, moving a file is achieved by selecting it and dragging it into a relevant folder or filing cabinet. The advantage of using a metaphor is that the user can identify with the environment presented on the screen. Having a metaphor allows users to predict the outcome of their actions more easily.

Note that the metaphor can break down, however. What is the real-world equivalent of formatting a disk? Is there a direct analogy for the concept of 'undo'? Think of some more examples yourself.

RECOMMENDED READING

D. A. Norman, *The Psychology of Everyday Things*, Basic Books, 1988. (Republished as *The Design of Everyday Things* by Penguin, 1991.)
A classic text, which discusses psychological issues in designing everyday objects and addresses why such objects are often so difficult to use. Discusses the execution–evaluation cycle. Very readable and entertaining. See also his more recent books *Turn Signals are the Facial Expressions of Automobiles* [267], *Things That Make Us Smart* [268] and *The Invisible Computer* [269].

R. W. Bailey, *Human Performance Engineering: A Guide for System Designers*, Prentice Hall, 1982.
Detailed coverage of human factors and ergonomics issues, with plenty of examples.

G. Salvendy, *Handbook of Human Factors and Ergonomics*, John Wiley, 1997.
Comprehensive collection of contributions from experts in human factors and ergonomics.

M. Helander, editor, *Handbook of Human–Computer Interaction. Part II: User Interface Design*, North-Holland, 1988.
Comprehensive coverage of interface styles.

J. Raskin, *The Humane Interface: New Directions for Designing Interactive Systems*, Addison Wesley, 2000.
Jef Raskin was one of the central designers of the original Mac user interface. This book gives a personal, deep and practical examination of many issues of interaction and its application in user interface design.

M. Blythe, A. Monk, K. Overbeeke and P. Wright, editors, *Funology: From Usability to Enjoyment*, Kluwer, 2003.
This is an edited book with chapters covering many areas of user experience. It includes an extensive review of theory from many disciplines from psychology to literary theory and chapters giving design frameworks based on these. The theoretical and design base is grounded by many examples and case studies including a detailed analysis of virtual crackers.

4

Paradigms

OVERVIEW

- Examples of effective strategies for building interactive systems provide paradigms for designing usable interactive systems.
- The evolution of these usability paradigms also provides a good perspective on the history of interactive computing.
- These paradigms range from the introduction of time-sharing computers, through the WIMP and web, to ubiquitous and context-aware computing.

4.1 INTRODUCTION

As we noted in Chapter 3, the primary objective of an interactive system is to allow the user to achieve particular goals in some application domain, that is, the interactive system must be usable. The designer of an interactive system, then, is posed with two open questions:

1. How can an interactive system be developed to ensure its usability?
2. How can the usability of an interactive system be demonstrated or measured?

One approach to answering these questions is by means of example, in which successful interactive systems are commonly believed to enhance usability and, therefore, serve as *paradigms* for the development of future products.

We believe that we now build interactive systems that are more usable than those built in the past. We also believe that there is considerable room for improvement in designing more usable systems in the future. As discussed in Chapter 2, the great advances in computer technology have increased the power of machines and enhanced the bandwidth of communication between humans and computers. The impact of technology alone, however, is not sufficient to enhance its usability. As our machines have become more powerful, the key to increased usability has come from the creative and considered application of the technology to accommodate and augment the power of the human. Paradigms for interaction have for the most part been dependent upon technological advances and their creative application to enhance interaction.

In this chapter, we investigate some of the principal historical advances in interactive designs. What is important to notice here is that the techniques and designs mentioned are recognized as major improvements in interaction, though it is sometimes hard to find a consensus for the reason behind the success. It is even harder to predict ahead what the new paradigms will be. Often new paradigms have arisen through exploratory designs that have then been seen, after the fact, to have created a new base point for future design.

We will discuss 15 different paradigms in this chapter. They do not provide mutually exclusive categories, as particular systems will often incorporate ideas from more than one of the following paradigms. In a way, this chapter serves as a history of interactive system development, though our emphasis is not so much on historical accuracy as on interactive innovation. We are concerned with the advances in interaction provided by each paradigm.

4.2 PARADIGMS FOR INTERACTION

4.2.1 Time sharing

In the 1940s and 1950s, the significant advances in computing consisted of new hardware technologies. Mechanical relays were replaced by vacuum electron tubes. Tubes were replaced by transistors, and transistors by integrated chips, all of which meant that the amount of sheer computing power was increasing by orders of magnitude. By the 1960s it was becoming apparent that the explosion of growth in computing power would be wasted if there was not an equivalent explosion of ideas about how to channel that power. One of the leading advocates of research into human-centered applications of computer technology was J. C. R. Licklider, who became the director of the Information Processing Techniques Office of the US Department of Defense's Advanced Research Projects Agency (ARPA). It was Licklider's goal to finance various research centers across the United States in order to encourage new ideas about how best to apply the burgeoning computing technology.

One of the major contributions to come out of this new emphasis in research was the concept of *time sharing*, in which a single computer could support multiple users. Previously, the human (or more accurately, the programmer) was restricted to batch sessions, in which complete jobs were submitted on punched cards or paper tape to an operator who would then run them individually on the computer. Time-sharing systems of the 1960s made programming a truly interactive venture and brought about a subculture of programmers known as 'hackers' – single-minded masters of detail who took pleasure in understanding complexity. Though the purpose of the first interactive time-sharing systems was simply to augment the programming capabilities of the early hackers, it marked a significant stage in computer applications for human use. Rather than rely on a model of interaction as a pre-planned activity that resulted in a complete set of instructions being laid out for the computer to follow, truly interactive exchange between programmer and computer was possible. The computer could now project itself as a dedicated partner with each individual user and the increased throughput of information between user and computer allowed the human to become a more reactive and spontaneous collaborator. Indeed, with the advent of time sharing, real human–computer interaction was now possible.

4.2.2 Video display units

As early as the mid-1950s researchers were experimenting with the possibility of presenting and manipulating information from a computer in the form of images on a video display unit (VDU). These display screens could provide a more suitable medium than a paper printout for presenting vast quantities of strategic information for rapid assimilation. The earliest applications of display screen images were developed in military applications, most notably the Semi-Automatic Ground Environment (SAGE) project of the US Air Force. It was not until 1962, however, when a young graduate student at the Massachusetts Institute of Technology (MIT), Ivan Sutherland, astonished the established computer science community with his *Sketchpad* program, that the capabilities of visual images were realized. As described in Howard Rheingold's history of computing book *Tools for Thought* [305]:

> Sketchpad allowed a computer operator to use the computer to create, very rapidly, sophisticated visual models on a display screen that resembled a television set. The visual patterns could be stored in the computer's memory like any other data, and could be manipulated by the computer's processor. . . . But Sketchpad was much more

than a tool for creating visual displays. It was a kind of simulation language that enabled computers to translate abstractions into perceptually concrete forms. And it was a model for totally new ways of operating computers; by changing something on the display screen, it was possible, via Sketchpad, to change something in the computer's memory.

Sketchpad demonstrated two important ideas. First, computers could be used for more than just data processing. They could extend the user's ability to abstract away from some levels of detail, visualizing and manipulating different representations of the same information. Those abstractions did not have to be limited to representations in terms of bit sequences deep within the recesses of computer memory. Rather, the abstractions could be made truly visual. To enhance human interaction, the information within the computer was made more amenable to human consumption. The computer was made to speak a more human language, instead of the human being forced to speak more like a computer. Secondly, Sutherland's efforts demonstrated how important the contribution of one creative mind (coupled with a dogged determination to see the idea through) could be to the entire history of computing.

4.2.3 Programming toolkits

Douglas Engelbart's ambition since the early 1950s was to use computer technology as a means of complementing human problem-solving activity. Engelbart's idea as a graduate student at the University of California at Berkeley was to use the computer to teach humans. This dream of naïve human users actually learning from a computer was a stark contrast to the prevailing attitude of his contemporaries that computers were a purposely complex technology that only the intellectually privileged were capable of manipulating. Engelbart's dedicated research team at the Stanford Research Institute in the 1960s worked towards achieving the manifesto set forth in an article published in 1963 [124]:

> By 'augmenting man's intellect' we mean increasing the capability of a man to approach a complex problem situation, gain comprehension to suit his particular needs, and to derive solutions to problems. . . . We refer to a way of life in an integrated domain where hunches, cut-and-try, intangibles, and the human 'feel for the situation' usefully coexist with powerful concepts, streamlined terminology and notation, sophisticated methods, and high-powered electronic aids.

Many of the ideas that Engelbart's team developed at the Augmentation Research Center – such as word processing and the mouse – only attained mass commercial success decades after their invention. A live demonstration of his oNLine System (NLS, also later known as NLS/Augment) was given in the autumn of 1968 at the Fall Joint Computer Conference in San Francisco before a captivated audience of computer sceptics. We are not so concerned here with the interaction techniques that were present in NLS, as many of those will be discussed later. What is important here is the method that Engelbart's team adopted in creating their very innovative and powerful interactive systems with the relatively impoverished technology of the 1960s.

Engelbart wrote of how humans attack complex intellectual problems like a carpenter who produces beautifully complicated pieces of woodwork with a good set of tools. The secret to producing computing equipment that aided human problem-solving ability was in providing the right *toolkit*. Taking this message to heart, his team of programmers concentrated on developing the set of programming tools they would require in order to build more complex interactive systems. The idea of building components of a computer system that will allow you to rebuild a more complex system is called bootstrapping and has been used to a great extent in all of computing. The power of programming toolkits is that small, well-understood components can be composed in fixed ways in order to create larger tools. Once these larger tools become understood, they can continue to be composed with other tools, and the process continues.

4.2.4 Personal computing

Programming toolkits provide a means for those with substantial computing skills to increase their productivity greatly. But Engelbart's vision was not exclusive to the computer literate. The decade of the 1970s saw the emergence of computing power aimed at the masses, computer literate or not. One of the first demonstrations that the powerful tools of the hacker could be made accessible to the computer novice was a graphics programming language for children called LOGO. The inventor, Seymour Papert, wanted to develop a language that was easy for children to use. He and his colleagues from MIT and elsewhere designed a computer-controlled mechanical turtle that dragged a pen along a surface to trace its path. A child could quite easily pretend they were 'inside' the turtle and direct it to trace out simple geometric shapes, such as a square or a circle. By typing in English phrases, such as `Go forward` or `Turn left`, the child/programmer could teach the turtle to draw more and more complicated figures. By adapting the graphical programming language to a model which children could understand and use, Papert demonstrated a valuable maxim for interactive system development – no matter how powerful a system may be, it will always be more powerful if it is easier to use.

Alan Kay was profoundly influenced by the work of both Engelbart and Papert. He realized that the power of a system such as NLS was only going to be successful if it was as accessible to novice users as was LOGO. In the early 1970s his view of the future of computing was embodied in small, powerful machines which were dedicated to single users, that is *personal computers*. Together with the founding team of researchers at the Xerox Palo Alto Research Center (PARC), Kay worked on incorporating a powerful and simple visually based programming environment, Smalltalk, for the personal computing hardware that was just becoming feasible. As technology progresses, it is now becoming more difficult to distinguish between what constitutes a personal computer, or workstation, and what constitutes a mainframe. Kay's vision in the mid-1970s of the ultimate handheld personal computer – he called it the Dynabook – outstrips even the technology we have available today [197].

4.2.5 Window systems and the WIMP interface

With the advent and immense commercial success of personal computing, the emphasis for increasing the usability of computing technology focussed on addressing the single user who engaged in a dialog with the computer in order to complete some work. Humans are able to think about more than one thing at a time, and in accomplishing some piece of work, they frequently interrupt their current train of thought to pursue some other related piece of work. A personal computer system which forces the user to progress in order through all of the tasks needed to achieve some objective, from beginning to end without any diversions, does not correspond to that standard working pattern. If the personal computer is to be an effective dialog partner, it must be as flexible in its ability to 'change the topic' as the human is.

But the ability to address the needs of a different user task is not the only requirement. Computer systems for the most part react to stimuli provided by the user, so they are quite amenable to a wandering dialog initiated by the user. As the user engages in more than one plan of activity over a stretch of time, it becomes difficult for him to maintain the status of the overlapping threads of activity. It is therefore necessary for the computer dialog partner to present the context of each thread of dialog so that the user can distinguish them.

One presentation mechanism for achieving this dialog partitioning is to separate physically the presentation of the different logical threads of user–computer conversation on the display device. The *window* is the common mechanism associated with these physically and logically separate display spaces. We discussed windowing systems in detail in Chapter 3.

Interaction based on windows, icons, menus and pointers – the WIMP interface – is now commonplace. These interaction devices first appeared in the commercial marketplace in April 1981, when Xerox Corporation introduced the 8010 Star Information System. But many of the interaction techniques underlying a windowing system were used in Engelbart's group in NLS and at Xerox PARC in the experimental precursor to Star, the Alto.

4.2.6 The metaphor

In developing the LOGO language to teach children, Papert used the metaphor of a turtle dragging its tail in the dirt. Children could quickly identify with the real-world phenomenon and that instant familiarity gave them an understanding of how they could create pictures. Metaphors are used quite successfully to teach new concepts in terms of ones which are already understood, as we saw when looking at analogy in Chapter 1. It is no surprise that this general teaching mechanism has been successful in introducing computer novices to relatively foreign interaction techniques. We have already seen how metaphors are used to describe the functionality of many interaction widgets, such as windows, menus, buttons and palettes. Tremendous commercial successes in computing have arisen directly from a judicious choice of metaphor. The Xerox Alto and Star were the first workstations based on the metaphor of the office desktop. The majority of the management tasks on a standard workstation have to do with file manipulation. Linking the set of tasks associated with file manipulation to the filing tasks in a typical office environment makes the actual computerized tasks easier to understand at first. The success of the desktop metaphor is unquestionable. Another good example in the personal computing domain is the widespread use of the spreadsheet metaphor for accounting and financial modeling.

Very few will debate the value of a good metaphor for increasing the initial familiarity between user and computer application. The danger of a metaphor is usually realized after the initial honeymoon period. When word processors were first introduced, they relied heavily on the typewriter metaphor. The keyboard of a computer closely resembles that of a standard typewriter, so it seems like a good metaphor from which to start. However, the behavior of a word processor is different from any typewriter. For example, the space key on a typewriter is passive, producing nothing on the piece of paper and just moving the guide further along the current line. For a typewriter, a space is not a character. However, for a word processor, the blank space *is* a character which must be inserted within a text just as any other character is inserted. So an experienced typist is not going to be able to predict correctly the behavior of pressing the spacebar on the keyboard by appealing to his experience with a typewriter. Whereas the typewriter metaphor is beneficial for providing a preliminary understanding of a word processor, the analogy is inadequate for promoting a full understanding of how the word processor works. In fact, the metaphor gets in the way of the user understanding the computer.

A similar problem arises with most metaphors. Although the desktop metaphor is initially appealing, it falls short in the computing world because there are no office equivalents for ejecting a floppy disk or printing a document. When designers try too hard to make the metaphor stick, the resulting system can be more confusing. Who thinks it is intuitive to drag the icon of a floppy disk to the wastebasket in order to eject it from the system? Ordinarily, the wastebasket is used to dispose of things that we never want to use again, which is why it works for deleting files. We must accept that some of the tasks we perform with a computer do not have real-world equivalents, or if they do, we cannot expect a single metaphor to account for all of them.

Another problem with a metaphor is the cultural bias that it portrays. With the growing internationalization of software, it should not be assumed that a metaphor will apply across national boundaries. A meaningless metaphor will only add another layer of complexity between the user and the system.

A more extreme example of metaphor occurs with *virtual reality* systems. In a VR system, the metaphor is not simply captured on a display screen. Rather, the user is also portrayed within the metaphor, literally creating an alternative, or virtual, reality. Any actions that the user performs are supposed to become more natural and so more movements of the user are interpreted, instead of just keypresses, button clicks and movements of an external pointing device. A VR system also needs to know the location and orientation of the user. Consequently, the user is often ‘rigged’ with special tracking devices so that the system can locate them and interpret their motion correctly.

4.2.7 Direct manipulation

In the early 1980s as the price of fast and high-quality graphics hardware was steadily decreasing, designers were beginning to see that their products were gaining popularity as their visual content increased. As long as the user–system dialog remained largely unidirectional – from user command to system command line prompt – computing was going to stay within the minority population of the hackers who revelled in the challenge of complexity. In a standard command line interface, the only way to get any feedback on the results of previous interaction is to know that you have to ask for it and to know how to ask for it. Rapid visual and audio *feedback* on a high-resolution display screen or through a high-quality sound system makes it possible to provide evaluative information for every executed user action.

Rapid feedback is just one feature of the interaction technique known as *direct manipulation*. Ben Shneiderman [320, 321] is attributed with coining this phrase in 1982 to describe the appeal of graphics-based interactive systems such as Sketchpad and the Xerox Alto and Star. He highlights the following features of a direct manipulation interface:

- visibility of the objects of interest
- incremental action at the interface with rapid feedback on all actions
- reversibility of all actions, so that users are encouraged to explore without severe penalties
- syntactic correctness of all actions, so that every user action is a legal operation
- replacement of complex command languages with actions to manipulate directly the visible objects (and, hence, the name direct manipulation).

The first real commercial success which demonstrated the inherent usability of direct manipulation interfaces for the general public was the Macintosh personal computer, introduced by Apple Computer, Inc. in 1984 after the relatively unsuccessful marketing attempt in the business community of the similar but more pricey Lisa computer. We discussed earlier how the desktop metaphor makes the computer domain of file management, usually described in terms of files and directories, easier to grasp by likening it to filing in the typical office environment, usually described in terms of documents and folders. The direct manipulation interface for the desktop metaphor requires that the documents and folders are made visible to the user as icons which represent the underlying files and directories. An operation such as moving a file from one directory to another is mirrored as an action on the visible document which is 'picked up and dragged' along the desktop from one folder to the next. In a command line interface to a filing system, it is normal that typographical errors in constructing the command line for a move operation would result in a syntactically incorrect command (for example, mistyping the file's name results in an error if you are fortunate enough not to spell accidentally the name of another file in the process). It is impossible to formulate a syntactically incorrect move operation with the pick-up-and-drag style of command. It is still possible for errors to occur at a deeper level, as the user might move a document to the wrong place, but it is relatively easy to detect and recover from those errors. While the document is dragged, continual visual feedback is provided, creating the illusion that the user is actually working in the world of the desktop and not just using the metaphor to help him understand.

Ed Hutchins, Jim Hollan and Donald Norman [187] provide a more psychological justification in terms of the *model-world metaphor* for the directness that the above example suggests. In Norman and Draper's collection of papers on user-centered design [270] they write:

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

In the model-world metaphor, the role of the interface is not so much one of mediating between the user and the underlying system. From the user's perspective, the interface *is* the system.

A consequence of the direct manipulation paradigm is that there is no longer a clear distinction between input and output. In the interaction framework in Chapter 3 we talked about a user articulating input expressions in some input language and observing the system-generated output expressions in some output language. In a direct manipulation system, the output expressions are used to formulate subsequent input expressions. The document icon is an output expression in the desktop metaphor, but that icon is used by the user to articulate the move operation. This aggregation of input and output is reflected in the programming toolkits, as widgets are not considered as input or output objects exclusively. Rather, widgets embody both input and output languages, so we consider them as *interaction objects*.

Somewhat related to the visualization provided by direct manipulation is the *WYSIWYG* paradigm, which stands for 'what you see is what you get'. What you see on a display screen, for example when you are using a word processor, is not the actual document that you will be producing in the end. Rather, it is a representation or rendering of what that final document will look like. The implication with a WYSIWYG interface is that the difference between the representation and the final product is minimal, and the user is easily able to visualize the final product from the computer's representation. So, in the word-processing example, you would be able to see what the overall layout of your document would be from its image on screen, minimizing any guesswork on your part to format the final printed copy.

With WYSIWYG interfaces, it is the simplicity and immediacy of the mapping between representation and final product that matters. In terms of the interaction framework, the observation of an output expression is made simple so that assessment of goal achievement is straightforward. But WYSIWYG is not a panacea for usability. What you see is all you get! In the case of a word processor, it is difficult to achieve more sophisticated page design if you must always see the results of the layout on screen. For example, suppose you want to include a picture in a document you are writing. You design the picture and then place it in the current draft of your document, positioning it at the top of the page on which it is first referenced. As you make changes to the paper, the position of the picture will change. If you still want it to appear at the top of a page, you will no doubt have to make adjustments to the document. It would be easier if you only had to include the picture once, with a directive that it should be positioned at the top of the printed page, whether or not it appears that way on screen. You might sacrifice the WYSIWYG principle in order to make it easier to incorporate such floatable objects in your documents.

Worked exercise

Discuss the ways in which a full-page word processor is or is not a direct manipulation interface for editing a document using Shneiderman's criteria. What features of a modern word processor break the metaphor of composition with pen (or typewriter) and paper?

Answer We will answer the first point by evaluating the word processor relative to the criteria for direct manipulation given by Shneiderman.

Visibility of the objects of interest

The most important objects of interest in a word processor are the words themselves. Indeed, the visibility of the text on a continual basis was one of the major usability advances in moving from line-oriented to display-oriented editors. Depending on the user's application, there may be other objects of interest in word processing that may or may not be visible. For example, are the margins for the text on screen similar to the ones which would eventually be printed? Is the spacing within a line and the line breaks similar? Are the different fonts and formatting characteristics of the text visible (without altering the spacing)? Expressed in this way, we can see the visibility criterion for direct manipulation as very similar to the criteria for a WYSIWYG interface.

Incremental action at the interface with rapid feedback on all actions

We expect from a word processor that characters appear in the text as we type them in at the keyboard, with little delay. If we are inserting text on a page, we might also expect that the format of the page adjust immediately to accommodate the new changes. Various word processors do this reformatting immediately, whereas with others changes in page breaks may take some time to be reflected. One of the other important actions which requires incremental and rapid feedback is movement of the window using the scroll button. If there is a significant delay between the input command to move the window down and the actual movement of the window on screen, it is quite possible that the user will 'overshoot' the target when using the scrollbar button.

Reversibility of all actions, so that users are encouraged to explore without severe penalties

Single-step undo commands in most word processors allow the user to recover from the last action performed. One problem with this is that the user must recognize the error before doing any other action. More sophisticated undo facilities allow the user to retrace back more than one command at a time. The kind of exploration this reversibility provides in a word processor is best evidenced with the ease of experimentation that is now available for formatting changes in a document (font types and sizes and margin changes). One problem with the ease of exploration is that emphasis may move to the look of a document rather than what the text actually says (style over content).

Syntactic correctness of all actions, so that every user action is a legal operation
WYSIWYG word processors usually provide menus and buttons which the user uses to articulate many commands. These interaction mechanisms serve to constrain the input language to allow only legal input from the user. Document markup systems, such as HTML and LaTeX, force the user to insert textual commands (which may be erroneously entered by the user) to achieve desired formatting effects.

Replacement of complex command languages with actions to manipulate directly the visible objects
The case for word processors is similar to that described above for syntactic correctness. In addition, operations on portions of text are achieved many times by allowing the user to highlight the text directly with a mouse (or arrow keys). Subsequent action on that text, such as moving it or copying it to somewhere else, can then be achieved more directly by allowing the user to 'drag' the selected text via the mouse to its new location.

To answer the second question concerning the drawback of the pen (or typewriter) metaphor for word processing, we refer to the discussion on metaphors in Section 4.2.6. The example there compares the functionality of the space key in typewriting versus word processing. For a typewriter, the space key is passive; it merely moves the insertion point one space to the right. In a word processor, the space key is active, as it inserts a character (the space character) into the document. The functionality of the typewriter space key is produced by the movement keys for the word processor (typically an arrow key pointing right to move forward within one line). In fact, much of the functionality that we have come to expect of a word processor is radically different from that expected of a typewriter, so much so that the typewriter as a metaphor for word processing is not all that instructive. In practice, modern typewriters have begun to borrow from word processors when defining their functionality!

4.2.8 Language versus action

Whereas it is true that direct manipulation interfaces make some tasks easier to perform correctly, it is equally true that some tasks are more difficult, if not impossible. Contrary to popular wisdom, it is not generally true that actions speak louder than words. The image we projected for direct manipulation was of the interface as a replacement for the underlying system as the world of interest to the user. Actions performed at the interface replace any need to understand their meaning at any deeper, system level. Another image is of the interface as the interlocutor or mediator between the user and the system. The user gives the interface instructions and it is then the responsibility of the interface to see that those instructions are carried out. The user–system communication is by means of indirect language instead of direct actions.

We can attach two meaningful interpretations to this language paradigm. The first requires that the user understands how the underlying system functions and the interface as interlocutor need not perform much translation. In fact, this interpretation of the language paradigm is similar to the kind of interaction which existed before direct manipulation interfaces were around. In a way, we have come full circle!

The second interpretation does not require the user to understand the underlying system's structure. The interface serves a more active role, as it must interpret between the intended operation as requested by the user and the possible system operations that must be invoked to satisfy that intent. Because it is more active, some people refer to the interface as an *agent* in these circumstances. We can see this kind of language paradigm at work in an information retrieval system. You may know what kind of information is in some internal system database, such as the UK highway code, but you would not know how that information is organized. If you had a question about speed limits on various roads, how would you ask? The answer in this case is that you would ask the question in whatever way it comes to mind, typing in a question such as, 'What are the speed limits on different roads?' You then leave it up to the interface agent to reinterpret your request as a legal query to the highway code database.

Whatever interpretation we attach to the language paradigm, it is clear that it has advantages and disadvantages when compared with the action paradigm implied by direct manipulation interfaces. In the action paradigm, it is often much easier to perform simple tasks without risk of certain classes of error. For example, recognizing and pointing to an object reduces the difficulty of identification and the possibility of misidentification. On the other hand, more complicated tasks are often rather tedious to perform in the action paradigm, as they require repeated execution of the same procedure with only minor modification. In the language paradigm, there is the possibility of describing a generic procedure once (for example, a looping construct which will perform a routine manipulation on all files in a directory) and then leaving it to be executed without further user intervention.

The action and language paradigms need not be completely separate. In the above example, we distinguished between the two paradigms by saying that we can describe generic and repeatable procedures in the language paradigm and not in the action paradigm. An interesting combination of the two occurs in *programming by example* when a user can perform some routine tasks in the action paradigm and the system records this as a generic procedure. In a sense, the system is interpreting the user's actions as a language script which it can then follow.

4.2.9 Hypertext

In 1945, Vannevar Bush, then the highest-ranking scientific administrator in the US war effort, published an article entitled 'As We May Think' in *The Atlantic Monthly*. Bush was in charge of over 6000 scientists who had greatly pushed back the frontiers of scientific knowledge during the Second World War. He recognized that a major drawback of these prolific research efforts was that it was becoming increasingly difficult to keep in touch with the growing body of scientific knowledge in the literature. In his opinion, the greatest advantages of this scientific revolution were to be gained by those individuals who were able to keep abreast of an ever-increasing flow of information. To that end, he described an innovative and futuristic information storage and retrieval apparatus – the *memex* – which was constructed with technology wholly existing in 1945 and aimed at increasing the human capacity to store and retrieve connected pieces of knowledge by mimicking our ability to create random associative links.

The memex was essentially a desk with the ability to produce and store a massive quantity of photographic copies of documented information. In addition to its huge storage capacity, the memex could keep track of links between parts of different documents. In this way, the stored information would resemble a vast interconnected mesh of data, similar to how many perceive information is stored in the human brain. In the context of scientific literature, where it is often very difficult to keep track of the origins and interrelations of the ever-growing body of research, a device which explicitly stored that information would be an invaluable asset.

We have already discussed some of the contributions of 'disciples' of Bush's vision – Douglas Engelbart and Alan Kay. One other follower was equally influenced by the ideas behind the memex, though his dreams have not yet materialized to the extent of Engelbart's and Kay's. Ted Nelson was another graduate student/dropout whose research agenda was forever transformed by the advent of the computer. An unsuccessful attempt to create a machine language equivalent of the memex on early 1960s computer hardware led Nelson on a lifelong quest to produce *Xanadu*, a potentially revolutionary worldwide publishing and information retrieval system based on the idea of interconnected, non-linear text and other media forms. A traditional paper is read from beginning to end, in a linear fashion. But within that text, there are often ideas or footnotes that urge the reader to digress into a richer topic. The linear format for information does not provide much support for this random and associated browsing task. What Bush's memex suggested was to preserve the non-linear browsing structure in the actual documentation. Nelson coined the phrase *hypertext* in the mid-1960s to reflect this non-linear text structure.

It was nearly two decades after Nelson coined the term that the first hypertext systems came into commercial use. In order to reflect the use of such non-linear and associative linking schemes for more than just the storage and retrieval of textual information, the term *hypermedia* (or *multimedia*) is used for non-linear storage of all forms of electronic media. We will discuss these systems in Part 4 of this book (see Chapter 21). Most of the riches won with the success of hypertext and hypermedia were not gained by Nelson, though his project Xanadu survives to this day.

4.2.10 Multi-modality

The majority of interactive systems still use the traditional keyboard and a pointing device, such as a mouse, for input and are restricted to a color display screen with some sound capabilities for output. Each of these input and output devices can be considered as communication channels for the system and they correspond to certain human communication channels, as we saw in Chapter 1. A *multi-modal* interactive system is a system that relies on the use of multiple human communication channels. Each different channel for the user is referred to as a modality of interaction. In this sense, all interactive systems can be considered multi-modal, for humans have always used their visual and haptic (touch) channels in manipulating a computer. In fact, we often use our audio channel to hear whether the computer is actually running properly.

However, genuine multi-modal systems rely to a greater extent on simultaneous use of multiple communication channels for both input and output. Humans quite naturally process information by simultaneous use of different channels. We point to someone and refer to them as 'you', and it is only by interpreting the simultaneous use of voice and touch that our directions are easily articulated and understood. Designers have wanted to mimic this flexibility in both articulation and observation by extending the input and output expressions an interactive system will support. So, for example, we can modify a gesture made with a pointing device by speaking, indicating what operation is to be performed on the selected object.

Multi-modal, multimedia and virtual reality systems form a large core of current research in interactive system design. These are discussed in more detail in Chapters 10, 20 and 21.

4.2.11 Computer-supported cooperative work

Another development in computing in the 1960s was the establishment of the first computer networks which allowed communication between separate machines. Personal computing was all about providing individuals with enough computing power so that they were liberated from dumb terminals which operated on a time-sharing system. It is interesting to note that as computer networks became widespread, individuals retained their powerful workstations but now wanted to reconnect themselves to the rest of the workstations in their immediate working environment, and even throughout the world! One result of this reconnection was the emergence of collaboration between individuals via the computer – called computer-supported cooperative work, or CSCW.

The main distinction between CSCW systems and interactive systems designed for a single user is that designers can no longer neglect the society within which any single user operates. CSCW systems are built to allow interaction between humans via the computer and so the needs of the many must be represented in the one product. A fine example of a CSCW system is electronic mail – *email* – yet another metaphor by which individuals at physically separate locations can communicate via electronic messages which work in a similar way to conventional postal systems. One user can compose a message and 'post' it to another user (specified by his electronic mail address). When the message arrives at the remote user's site, he is informed that a new message has arrived in his 'mailbox'. He can then read the message and respond as desired. Although email is modeled after conventional postal systems, its major advantage is that it is often much faster than the traditional system (jokingly referred to by email devotees as 'snail mail'). Communication turnarounds between sites across the world are in the order of minutes, as opposed to weeks.

Electronic mail is an instance of an asynchronous CSCW system because the participants in the electronic exchange do not have to be working at the same time in order for the mail to be delivered. The reason we use email is precisely because of its asynchronous characteristics. All we need to know is that the recipient will eventually receive the message. In contrast, it might be desirable for synchronous communication, which would require the simultaneous participation of sender and recipient, as in a phone conversation.

CSCW is a major emerging topic in current HCI research, and so we devote much more attention to it later in this book. CSCW systems built to support users working in groups are referred to as *groupware*. Chapter 19 discusses groupware systems in depth. In Chapter 14 the more general issues and theories arising from CSCW are discussed.

4.2.12 The world wide web

Probably the most significant recent development in interactive computing is the world wide web, often referred to as just the web, or WWW. The web is built on top of the internet, and offers an easy to use, predominantly graphical interface to information, hiding the underlying complexities of transmission protocols, addresses and remote access to data.

The internet (see Section 2.9) is simply a collection of computers, each linked by any sort of data connection, whether it be slow telephone line and modem or high-bandwidth optical connection. The computers of the internet all communicate using common data transmission protocols (*TCP/IP*) and addressing systems (*IP* addresses and *domain names*). This makes it possible for anyone to read anything from anywhere, in theory, if it conforms to the protocol. The web builds on this with its own layer of network protocol (http), a standard markup notation (such as HTML) for laying out pages of information and a global naming scheme (uniform resource locators or URLs). Web pages can contain text, color images, movies, sound and, most important, hypertext links to other web pages. Hypermedia documents can therefore be 'published' by anyone who has access to a computer connected to the internet.

The world wide web project was conceived in 1989 by Tim Berners-Lee, working at CERN, the European Particle Physics Laboratory at Geneva, as a means to enable the widespread distribution of scientific data generated at CERN and to share information between physicists worldwide. In 1991 the first text-based web browser was released. This was followed in early 1993 by several graphical web browsers, most significantly Mosaic developed by Marc Andreesen at the National Center for Supercomputer Applications (NCSA) at Champaign, Illinois. This was the defining moment at which the meteoric growth of the web began, rapidly growing to dominate internet traffic and change the public view of computing. Of all the 'heroes' of interactive computing named in this chapter, it is only Berners-Lee who has achieved widespread public fame.

Whilst the internet has been around since 1969, it did not become a major paradigm for interaction until the advent and ease of availability of well-designed graphical interfaces (browsers) for the web. These browsers allow users to access multimedia information easily, using only a mouse to point and click. This shift towards the integration of computation and communication is transparent to users; all they realize is that they can get the current version of published information practically instantly. In addition, the language used to create these multimedia documents is relatively simple, opening the opportunity of publishing information to any literate, and connected, person. However, there are important limitations of the web as a hypertext medium and in Chapter 21 we discuss some of the special design issues for the web. Interestingly, the web did not provide any technological breakthroughs; all the required functionality previously existed, such as transmission protocols, distributed file systems, hypertext and so on. The impact has been due to the ease of use of both the browsers and HTML, and the fact that *critical mass* (see Chapter 13) was established, first in academic circles, and then rapidly expanded into the leisure and business domains. The burgeoning interest led to service providers, those providing connections to the internet, to make it cheap to connect, and a whole new subculture was born.

Currently, the web is one of the major reasons that new users are connecting to the internet (probably even buying computers in the first place), and is rapidly becoming a major activity for people both at work and for leisure. It is much more a social phenomenon than anything else, with users attracted to the idea that computers are now boxes that connect them with interesting people and exciting places to go, rather than soulless cases that deny social contact. Computing often used to be seen as an anti-social activity; the web has challenged this by offering a 'global village' with free access to information and a virtual social environment. Web culture has emphasized liberality and (at least in principle) equality regardless of gender, race and disability. In practice, the demographics of web users are only now coming close to equal proportions in terms of gender, and, although internet use is increasing globally, the vast majority of websites are still hosted in the United States. Indeed, the web is now big business; corporate images and e-commerce may soon dominate the individual and often zany aspects of the web.

4.2.13 Agent-based interfaces

In the human world agents are people who work on someone's behalf: estate agents buy and sell property for their customers, literary agents find publishers for authors, travel agents book hotels and air tickets for tourists and secret agents obtain information (secretly) for their governments. Software agents likewise act on behalf of users within the electronic world. Examples include email agents which filter your mail for you and web crawlers which search the world wide web for documents you might find interesting. Agents can perform repetitive tasks, watch and respond to events when the user is not present and even learn from the user's own actions.

Some agents simply do what they are told. For example, many email systems allow you to specify filters, simple *if then* rules, which determine the action to perform on certain kinds of mail message:

If Sender: is bank manager
Then Urgency: is high

A major problem with such agents is developing a suitable language between human and agent which allows the user to express intentions. This is especially important when the agent is going to act in the user's absence. In this case, the user may not receive feedback of any mistake until long after the effects have become irreversible; hence the instructions have to be correct, and believed to be correct.

Other agents use artificial intelligence techniques to learn based on the user's actions. An early example of this was Eager [83]. Eager watches users while they work on simple HyperCard applications. When it notices that the user is repeating similar actions a small icon appears (a smiling cat!), suggesting the next action. The user is free either to accept the suggestion or to ignore it. When the user is satisfied that Eager knows what it is doing, it can be instructed to perform all the remaining actions in a sequence.

Eager is also an example of an agent, which has a clear *embodiment*, that is, there is a representation of Eager (the cat icon) in the interface. In contrast, consider Microsoft Excel which incorporates some intelligence in its sum (Σ) function. If the current cell is directly below a column of numbers, or if there is a series of numbers to the left of the current cell, the sum range defaults to be the appropriate cells. It is also clever about columns of numbers with subtotals so that they are not included twice in the overall total. As around 80% of all spreadsheet formulae are simple sums this is a very useful feature. However, the intelligence in this is not embodied, it is diffuse, somewhere in 'the system'. Although embodiment is not essential to an agent-based system it is one of the key features which enable users to determine where autonomy and intelligence may lie, and also which parts are stable [107].

We have already discussed the relationship between language and action paradigms in human–computer interaction. To some extent agent-based systems include aspects of both. Old command-based systems acted as intermediaries: you asked them to do something, they did what you wanted (if you were lucky), and then reported the results back to you. In contrast, direct manipulation emphasizes the user's own actions, possibly augmented by tools, on the electronic world. Agents act on the user's behalf, possibly, but not necessarily, instructed in a linguistic fashion. But unlike the original intermediary paradigm, an agent is typically acting within a world the user could also act upon. The difference is rather like that between a traditional shopkeeper who brings items to you as opposed to a shop assistant in a supermarket who helps you as you browse amongst the aisles. The latter does not prevent you from selecting your own items from the shelves, but aids you when asked.

In fact, the proponents of direct manipulation and agent-based systems do not see the paradigms as being quite as complementary as we have described them above. Although amicable, the positions on each side are quite entrenched.

4.2.14 Ubiquitous computing

Where does computing happen, and more importantly, where do we as users go to interact with a computer? The past 50 years of interactive computing show that we still think of computers as being confined to a box on a desk or in an office or lab. The actual form of the physical interface has been transformed from a noisy teletype terminal to a large, graphical display with a WIMP or natural language interface, but in all cases the user knows where the computer is and must walk over to it to begin interacting with it.

In the late 1980s, a group of researchers at Xerox PARC, led by Mark Weiser, initiated a research program with the goal of moving human–computer interaction away from the desktop and out into our everyday lives. Weiser observed:

> The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.

These words have inspired a new generation of researchers in the area of *ubiquitous computing* [369, 370]. Another popular term for this emerging paradigm is *pervasive* computing, first coined by IBM. The intention is to create a computing infrastructure that permeates our physical environment so much that we do not notice the computer any longer. A good analogy for the vision of ubiquitous computing is the electric motor. When the electric motor was first introduced, it was large, loud and very noticeable. Today, the average household contains so many electric motors that we hardly ever notice them anymore. Their utility led to ubiquity and, hence, invisibility.

How long in the future will it be before we no longer notice the interactive computer? To some extent, this is already happening, since many everyday items, such as watches, microwaves or automobiles, contain many microprocessors that we don't directly notice. But, to a large extent, the vision of Weiser, in which the computer is hardly ever noticed, is a long way off.

To pursue the analogy with the electric motor a little further, one of the motor's characteristics is that it comes in many sizes. Each size is suited to a particular use. Weiser thought that it was also important to think of computing technology in different sizes. The original work at PARC looked at three different scales of computing: the yard, the foot and the inch. In the middle of the scale, a foot-sized computer is much like the personal computers we are familiar with today. Its size is suitable for every individual to have one, perhaps on their desk or perhaps in their bedroom or in their briefcase. A yard-sized computer, on the other hand, is so large that it would be suitable for wide open public spaces, and would be shared by a group of people. Perhaps there would be one of these in every home, or in a public hallway or auditorium. On the opposite side of the scale, an inch-sized computer would be a truly personal computing device that could fit in the palm of a hand. Everyone would have a number of these at their disposal, and they would be as prevalent and unremarkable as a pen or a pad of sticky notes.

There is an increasing number of examples of computing devices at these different scales. At the foot scale, laptop computers are, of course, everywhere, but more interesting examples of computing at this scale are commercially available *tablet computers* or research prototypes, such as an interactive storybook (see Figure 4.1). At the yard scale, there are various forms of high-resolution large screens and projected displays as we discussed in Chapter 2 (Section 2.4.3). These are still mainly used as output-only devices showing presentations or fixed messages, but there is

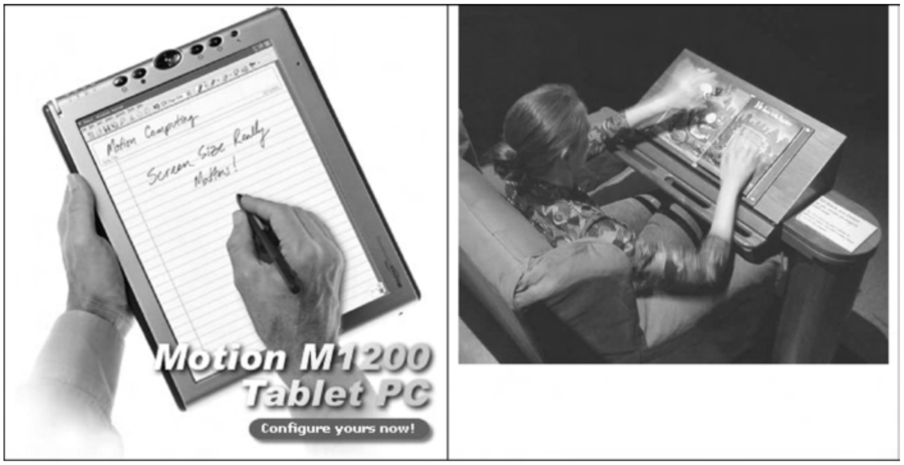


Figure 4.1 Examples of computing devices at the foot scale. On the left is a tablet computer – a Tablet PC from MotionComputing (Source: Motion Computing, Inc.). On the right is a research prototype, the Listen Reader, an interactive storybook developed at Palo Alto Research Center (picture courtesy Palo Alto Research Center)

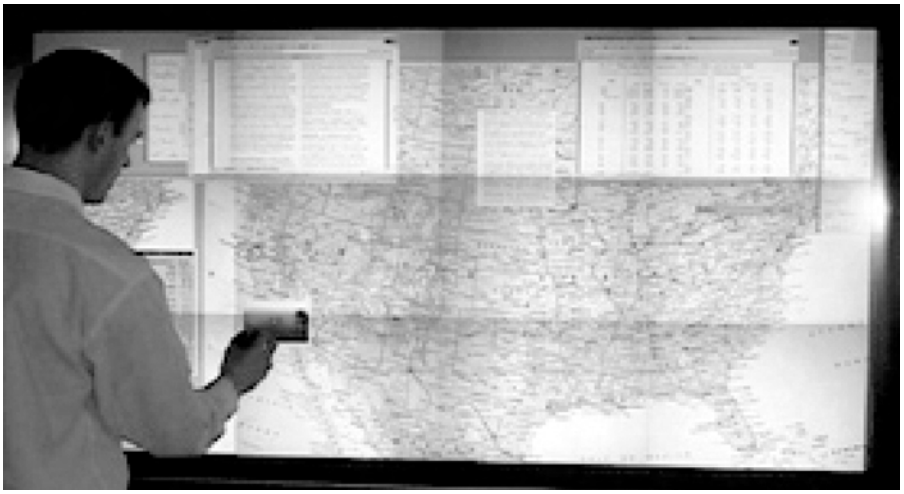


Figure 4.2 The Stanford Interactive Mural, an example of a yard-scale interactive display surface created by tiling multiple lower-resolution projectors. Picture courtesy François Guimbretière

increasing use of more interactive shared public displays, such as the Stanford Interactive Mural shown in Figure 4.2. At the inch scale, there are many examples, from powerful, pocket-sized *personal organizers* or *personal digital assistants* (PDAs) to even smaller cellular phones or pagers, and many pocket electronic devices such as electronic dictionaries and translators (see Figure 4.3). There are even badges whose position can be automatically tracked.

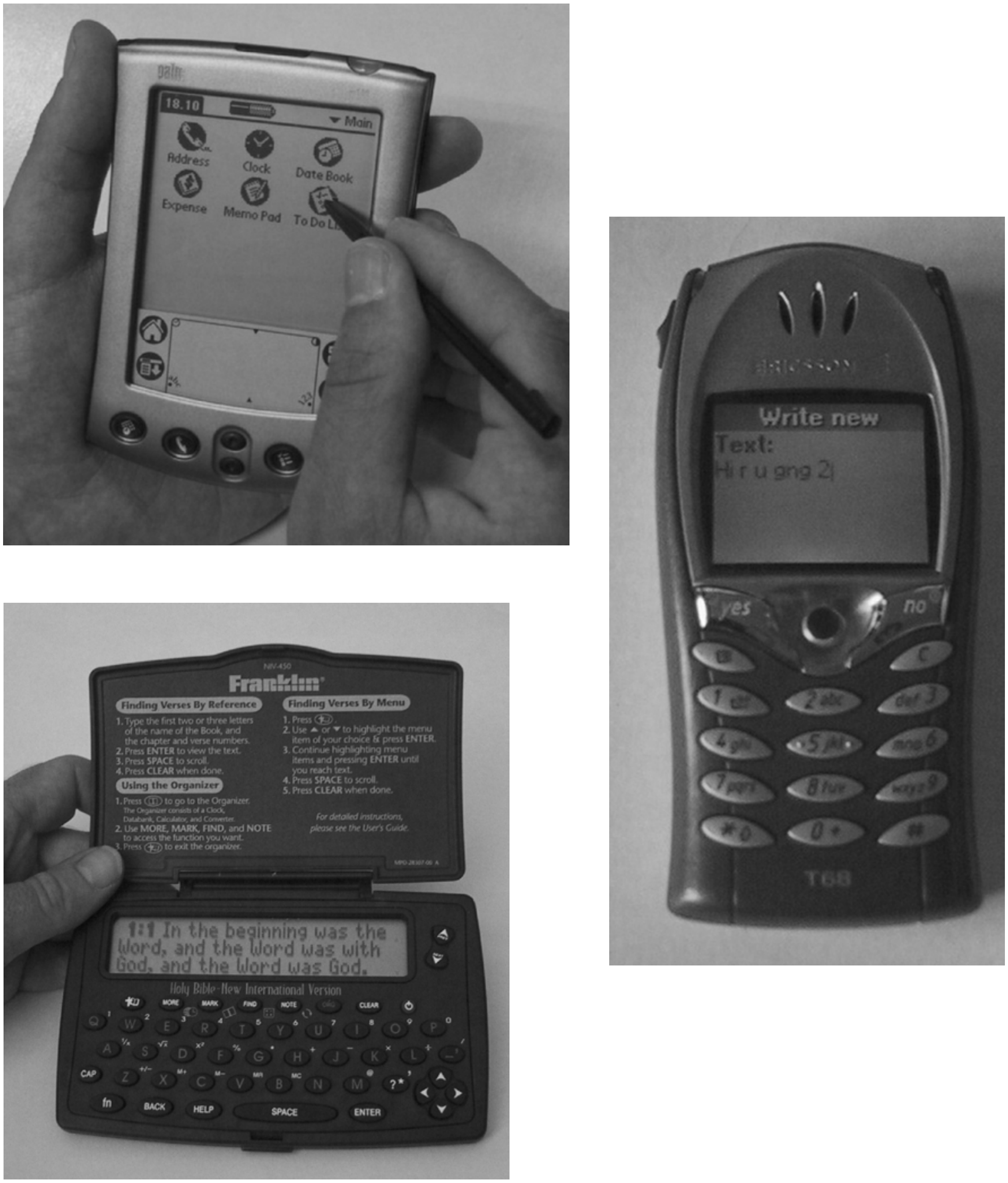

18.10
Main
Address
Clock
Date Book
Expense
Memo Pad
To Do List
Write new
Text:
yes
no
T68
Franklin
1:1 In the beginning was the Word, and the Word was with God, and the Word was God.
Holy Bible-New International Version

Figure 4.3 Example inch-scale devices. From left to right, a PDA, a mobile phone and pocket-sized electronic bible. Source: Top left photograph by Alan Dix (Palm Pilot Series V), bottom left photograph by Alan Dix with permission from Franklin Electronic Publishers, photograph right by Alan Dix (Ericsson phone)

This influx of diverse computing devices represents the *third wave of computing*, in which the ratio of computers to human drastically changes. In the first wave of computing, one large mainframe computer served many people. In the second wave, the PC revolution, computing devices roughly equalled the number of people using them. In the third wave, the devices far outnumber the people. It is precisely because of the large ratio of devices to people that Weiser and others note the importance of minimizing the attention demands of any single device.

Many different technologies are converging to make the dream of ubiquitous computing possible. These technologies include wireless networking, voice recognition, camera and vision systems, pen-based computing and positioning systems, to name a few. What all of these technologies provide is the ability to move the computer user away from a desktop, allow interaction in a number of modes (voice, gesture, handwriting) in addition to a keyboard, and make information about the user (through vision, speech recognition or positioning information) available to a computational device that may be far removed from the actual user.

Ubiquitous computing is not simply about nifty gadgets, it is what can be done with those gadgets. As Weiser pointed out, it is the applications that make ubiquitous computing revolutionary. In Chapter 20, we discuss some examples of the applications that ubiquitous computing makes possible, including the way this is becoming part of everyday life in places as diverse as the home, the car and even our own bodies. The vision of ubiquitous computing – first expressed by Weiser and grounded in experimental work done at Xerox PARC – is now starting to become reality.

4.2.15 Sensor-based and context-aware interaction

The yard-scale, foot-scale and inch-scale computers are all still clearly embodied devices with which we interact, whether or not we consider them 'computers'. There are an increasing number of proposed and existing technologies that embed computation even deeper, but unobtrusively, into day-to-day life. Weiser's dream was computers that 'permeate our physical environment so much that we do not notice the computers anymore', and the term ubiquitous computing encompasses a wide range from mobile devices to more pervasive environments.

We can consider the extreme situation in which the user is totally unaware of interaction taking place. Information can be gathered from sensors in the environment (pressure mats, ultrasonic movement detectors, weight sensors, video cameras), in our information world (web pages visited, times online, books purchased online), and even from our own bodies (heart rate, skin temperature, brain signals). This information is then used by systems that make inferences about our past patterns and current *context* in order to modify the explicit interfaces we deal with (e.g., modify default menu options) or to do things in the background (e.g., adjust the air conditioning).

We already encounter examples of this: lights that turn on when we enter a room, suggestions made for additional purchases at online bookstores, automatic doors and washbasins. For elderly and disabled people, assistive technologies already embody quite radical aspects of this. However, the vision of many is a world in which the whole environment is empowered to sense and even understand the context of activities within it.

Previous interactive computation has focussed on the user explicitly telling the computer exactly what to do and the computer doing what it is told. In *context-aware computing* the interaction is more implicit. The computer, or more accurately the sensor-enhanced environment, is using heuristics and other semi-intelligent means to predict what would be useful for the user. The data used for this inference and the inference itself are both fuzzy, probabilistic and uncertain. Automatically sensing context is, and will likely always remain, an imperfect activity, so it is important that the actions resulting from these 'intelligent' predictions be made with caution. Context-aware applications should follow the principles of *appropriate intelligence*:

1. Be right as often as possible, and useful when acting on these correct predictions.
2. Do not cause inordinate problems in the event of an action resulting from a wrong prediction.

The failure of 'intelligent' systems in the past resulted from following the first principle, but not the second. These new applications, which impinge so closely on our everyday lives, demand that the second principle of appropriate intelligence is upheld. (There is more on using intelligence in interfaces on the book website at /e3/online/intelligence/)

Arguably this is a more radical paradigm shift than any other since the introduction of interactive computing itself. Whereas ubiquitous computing challenges the idea of *where* computers are and how apparent they are to us, context-aware computing challenges *what it means to interact* with a computer. It is as if we have come full circle from the early days of computing. Large mainframes were placed in isolation from the principle users (programmers) and interaction was usually done through an intermediary operator. Half a century later, the implicit nature of interaction implied by sensing creates a human–computer relationship that becomes so seamless there is no conscious interaction at all.

This shift is so radical that one could even say it does not belong in this chapter about paradigms for interaction! In fact, this shift is so dramatic that it is unclear whether the basic models of interaction that have proved universal across technologies, for example Norman's execution–evaluation cycle (Chapter 3, Section 3.2.2), are applicable at all. We will return to this issue in Chapter 18.

4.3 SUMMARY

In this chapter, we have discussed paradigms that promote the usability of interactive systems. We have seen that the history of computing is full of examples of creative insight into how the interaction between humans and computers can be enhanced. While we expect never to replace the input of creativity in interactive system design, we still want to maximize the benefit of one good idea by repeating its benefit in many other designs. The problem with these paradigms is that they are rarely well defined. It is not always clear how they support a user in accomplishing some tasks. As a result, it is entirely possible that repeated use of some paradigm will not result in the design of a more usable system. The derivation of principles and theoretical models for interaction has often arisen out of a need to explain why a paradigm is successful and when it might not be. Principles can provide the repeatability that paradigms in themselves cannot provide. However, in defining these principles, it is all too easy to provide general and abstract definitions that are not very helpful to the designer. Therefore, the future of interactive system design relies on a complementary approach. The creativity that gives rise to new paradigms should be strengthened by the development of a theory that provides principles to support the paradigm in its repeated application. We will consider such principles and design rules in detail in Chapter 7 and more theoretical perspectives in Part 3.

EXERCISES

4.1. Choose one of the people mentioned in this chapter, or another important figure in the history of HCI, and create a web page biography on them. Try to get at least one picture of your subject, and find out about their life and work, with particular reference to their contribution to HCI.

4.2. Choose one paradigm of interaction and find three specific examples of it, not included in this chapter. Compare these three – can you identify any general principles of interaction that are embodied in each of your examples (see Chapter 7 for example principles)?

4.3. What new paradigms do you think may be significant in the future of interactive computing?

4.4. A truly ubiquitous computing experience would require the spread of computational capabilities literally everywhere. Another way to achieve ubiquity is to carry all of your computational needs with you everywhere, all the time. The field of *wearable computing* explores this interaction paradigm. How do you think the first-person emphasis of wearable computing compares with the third-person, or environmental, emphasis of ubiquitous computing? What impact would there be on context-aware computing if all of the sensors were attached to the individual instead of embedded in the environment?

RECOMMENDED READING

H. Rheingold, *Tools for Thought*, Prentice Hall, 1985.
An easy to read history of computing, with particular emphasis on developments in interactive systems. Much of the historical perspective of this chapter was influenced by this book.

T. Berners-Lee, *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web*, Harper, 2000.
The history of the world wide web as told by its inventor Tim Berners-Lee makes interesting and enlightening reading.

M. M. Waldrop, *The Dream Machine: J.C.R. Licklider and the Revolution That Made Computing Personal*, Penguin, 2002.
A readable biography of Licklider and his pioneering work in making computing available to all.

T. Bardini, *Bootstrapping: Douglas Englebart, Coevolution and the Origins of Personal Computing (Writing Science)*, Stanford University Press, 2000.
Biography of another great pioneer of interactive computing, Doug Engelbart and his work at Stanford.

J. M. Nyce and P. Kahn, editors, and V. Bush, *From Memex to Hypertext: Vannevar Bush and the Mind's Machine*, Academic Press, 1992.
An edited collection of Bush's writing about Memex, together with essays from computing historians.

M. Weiser, Some computer science issues in ubiquitous computing, *Communications of the ACM*, Vol. 36, No. 7, pp. 75–84, July 1993.
Classic article on issues relating to ubiquitous computing, including hardware, interaction software, networks, applications and methods.

A. Dix, T. Rodden, N. Davies, J. Trevor, A. Friday and K. Palfreyman, Exploiting space and location as a design framework for interactive mobile systems, *ACM Transactions on Computer–Human Interaction (TOCHI)*, Vol. 7, No. 3, pp. 285–321, September 2000.
Explores the space of context sensitive designs focussed especially on location.

There are also a number of special journal issues on ubiquitous, sensor-based and context-aware computing in *ACM TOCHI*, *Human–Computer Interaction* and *Communications of the ACM* and dedicated journals such as *IEEE Pervasive* and *Personal Technologies Journal*.

PART

2

Design process

In this part, we concentrate on how design practice addresses the critical feature of an interactive system – usability from the human perspective. The chapters in this part promote the purposeful design of more usable interactive systems. We begin in Chapter 5 by introducing the key elements in the interaction design process. These elements are then expanded in later chapters.

Chapter 6 discusses the design process in more detail, specifically focussing on the place of user-centered design within a software engineering framework. Chapter 7 highlights the range of design rules that can help us to specify usable interactive systems, including abstract principles, guidelines and other design representations.

In Chapter 8, we provide an overview of implementation support for the programmer of an interactive system. Chapter 9 is concerned with the techniques used to evaluate the interactive system to see if it satisfies user needs. Chapter 10 discusses how to design a system to be universally accessible, regardless of age, gender, cultural background or ability. In Chapter 11 we discuss the provision of user support in the form of help systems and documentation.

5 INTERACTION DESIGN BASICS

OVERVIEW

Interaction design is about creating interventions in often complex situations using technology of many kinds including PC software, the web and physical devices.

- Design involves:
  - achieving goals within constraints and trade-off between these
  - understanding the raw materials: computer and human
  - accepting limitations of humans and of design.
- The design process has several stages and is iterative and never complete.
- Interaction starts with getting to know the users and their context:
  - finding out who they are and what they are like . . . probably not like you!
  - talking to them, watching them.
- Scenarios are rich design stories, which can be used and reused throughout design:
  - they help us see what users will want to do
  - they give a step-by-step walkthrough of users' interactions: including what they see, do and are thinking.
- Users need to find their way around a system. This involves:
  - helping users know where they are, where they have been and what they can do next
  - creating overall structures that are easy to understand and fit the users' needs
  - designing comprehensible screens and control panels.
- Complexity of design means we don't get it right first time:
  - so we need iteration and prototypes to try out and evaluate
  - but iteration can get trapped in *local maxima*, designs that have no simple improvements, but are not good
  - theory and models can help give good start points.

5.1 INTRODUCTION

Some of HCI is focussed on understanding: the academic study of the way people interact with technology. However, a large part of HCI is about doing things and making things – design.

In this chapter we will think about interaction design. Note that we are not just thinking about the design of interactive systems, but about the interaction itself. An office has just got a new electric stapler. It is connected to the mains electricity and is hard to move around, so when you want to staple papers together you go to the stapler. In the past when someone wanted to staple things they would take the stapler to their desk and keep it until someone else wanted it. You might write a letter, print it, staple it, write the next letter, staple it, and so on. Now you have to take the letters to be stapled across the office, so instead you write–print, write–print until you have a pile of things to staple and then take them across. The stapler influences the whole pattern of interaction.

So, interaction design is not just about the artifact that is produced, whether a physical device or a computer program, but about understanding and choosing how that is going to affect the way people work. Furthermore, the artifacts we give to people are not just these devices and programs, but also manuals, tutorials, online help systems. In some cases we may realize that no additional system is required at all, we may simply suggest a different way of using existing tools.

Because of this it may be better not to think of designing a system, or an artifact, but to think instead about *designing interventions*. The product of a design exercise is that we intervene to change the situation as it is; we hope, of course, changing it for the better!

In the next section we will ask 'what is design?' which sets the spirit for the rest of the chapter. Section 5.3 looks at the design process as a whole and this gives a framework for the following sections. Section 5.4 looks at aspects of the requirements-gathering phase of design focussed on getting to know and understand the user. This is followed in Section 5.5 by a look at scenarios, which are a way of recording existing situations and examining proposed designs. We then look at the details of designing the overall application structure in Section 5.6 and individual screen design in Section 5.7. Because design is never perfect first time (or ever!), most interaction design involves several cycles of prototyping and evaluation. The chapter ends with an examination of the limits of this and why this emphasizes the importance of deep knowledge of more general theories and models of interaction.

This chapter also functions as an introduction to much of Part 2 and Part 3 of this book. In particular, Section 5.3 puts many of the succeeding chapters into the context of the overall design process. Many of the individual sections of this chapter give early views, or simple techniques, for issues and areas dealt with in detail later in the book.

5.2 WHAT IS DESIGN?

So what is design? A simple definition is:

achieving goals within constraints

This does not capture everything about design, but helps to focus us on certain things:

Goals What is the purpose of the design we are intending to produce? Who is it for? Why do they want it? For example, if we are designing a wireless personal movie player, we may think about young affluent users wanting to watch the latest movies whilst on the move and download free copies, and perhaps wanting to share the experience with a few friends.

Constraints What materials must we use? What standards must we adopt? How much can it cost? How much time do we have to develop it? Are there health and safety issues? In the case of the personal movie player: does it have to withstand rain? Must we use existing video standards to download movies? Do we need to build in copyright protection?

Of course, we cannot always achieve all our goals within the constraints. So perhaps one of the most important things about design is:

Trade-off Choosing which goals or constraints can be relaxed so that others can be met. For example, we might find that an eye-mounted video display, a bit like those used in virtual reality, would give the most stable image whilst walking along. However, this would not allow you to show friends, and might be dangerous if you were watching a gripping part of the movie as you crossed the road.

Often the most exciting moments in design are when you get a radically different idea that allows you to satisfy several apparently contradictory constraints. However, the more common skill needed in design is to accept the conflict and choose the most appropriate trade-off. The temptation is to focus on one or other goal and optimize for this, then tweak the design to make it just satisfy the constraints and other goals. Instead, the best designs are where the designer understands the trade-offs and the factors affecting them. Paradoxically, if you focus on the trade-off itself the more radical solutions also become more apparent.

5.2.1 The golden rule of design

Part of the understanding we need is about the circumstances and context of the particular design problem. We will return to this later in the chapter. However, there are also more generic concepts to understand. The designs we produce may be different, but often the raw materials are the same. This leads us to the *golden rule of design*:

understand your materials

In the case of a physical design this is obvious. Look at a chair with a steel frame and one with a wooden frame. They are very different: often the steel frames are tubular or thin L or H section steel. In contrast wooden chairs have thicker solid legs. If you made a wooden chair using the design for a metal one it would break; if you made the metal one in the design for the wooden one it would be too heavy to move.

For Human–Computer Interaction the obvious materials are the human and the computer. That is we must:

- understand *computers*
  - limitations, capacities, tools, platforms
- understand *people*
  - psychological, social aspects, human error.

Of course, this is exactly the focus of Chapters 1 and 2. This is why they came first; we must understand the fundamental materials of human–computer interaction in order to design. In Chapters 3 and 4 we also looked at the nature of *interaction* itself. This is equally important in other design areas. For example, the way you fit seats and windows into an airplane's hull affects the safety and strength of the aircraft as a whole.

5.2.2 To err is human

It might sound demeaning to regard people as 'materials', possibly even dehumanizing. In fact, the opposite is the case: physical materials are treated better in most designs than people. This is particularly obvious when it comes to failures.

The news headlines: an aircrash claims a hundred lives; an industrial accident causes millions of pounds' worth of damage; the discovery of systematic mistreatment leads to thousands of patients being recalled to hospital. Some months later the public inquiries conclude: human error in the operation of technical instruments. The phrase 'human error' is taken to mean 'operator error', but more often than not the disaster is inherent in the design or installation of the human interface. Bad interfaces are slow or error-prone to use. Bad interfaces cost money and cost lives.

People make mistakes. This is not 'human error', an excuse to hide behind in accident reports, it is human nature. We are not infallible consistent creatures, but often make slips, errors and omissions. A concrete lintel breaks and a building collapses. Do the headlines read 'lintel error'? No. It is the nature of concrete lintels to break if they are put under stress and it is the responsibility of architect and engineer to ensure that a building only puts acceptable stress on the lintel. Similarly it is the nature of humans to make mistakes, and systems should be designed to reduce the likelihood of those mistakes and to minimize the consequences when mistakes happen.

Often when an aspect of an interface is obscure and unclear, the response is to add another line in the manual. People are remarkably adaptable and, unlike concrete lintels, can get 'stronger', but better training and documentation (although necessary) are not a panacea. Under stress, arcane or inconsistent interfaces will lead to errors.

If you design using a physical material, you need to understand how and where failures would occur and strengthen the construction, build in safety features or redundancy. Similarly, if you treat the human with as much consideration as a piece of steel or concrete, it is obvious that you need to understand the way human failures occur and build the rest of the interface accordingly.

5.2.3 The central message – the user

In this book you will find information on basic psychology, on particular technologies, on methods and models. However, there is one factor that outweighs all this knowledge. It is about attitude. Often it is said that the success of the various methods used in HCI lies not in how good they are, but in that they simply focus the mind of the designer on the user.

This is the core of interaction design: put the user first, keep the user in the center and remember the user at the end.

5.3 THE PROCESS OF DESIGN

Often HCI professionals complain that they are called in too late. A system has been designed and built, and only when it proves unusable do they think to ask how to do it right! In other companies usability is seen as equivalent to testing – checking whether people can use it and fixing problems, rather than making sure they can from the beginning. In the best companies, however, usability is designed in from the start.

In Chapter 6 we will look in detail at the software development process and how HCI fits within it. Here we'll take a simplified view of four main phases plus an iteration loop, focussed on the design of interaction (Figure 5.1).

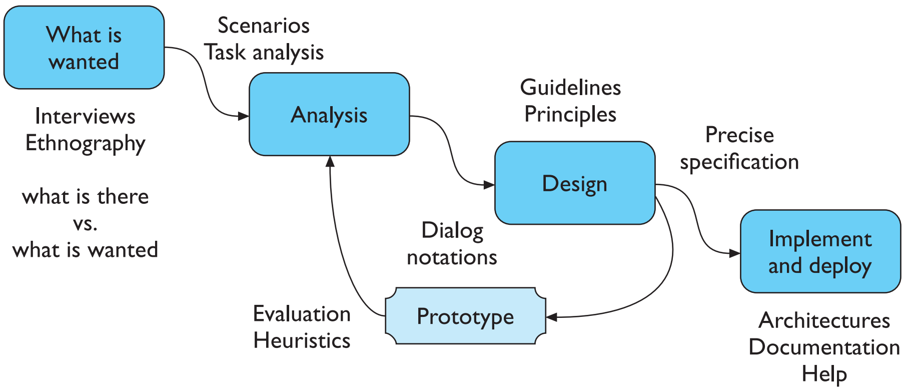


Figure 5.1 Interaction design process

Requirements – what is wanted The first stage is establishing what exactly is needed. As a precursor to this it is usually necessary to find out what is currently happening. For example, how do people currently watch movies? What sort of personal appliances do they currently use?

There are a number of techniques used for this in HCI: interviewing people, videotaping them, looking at the documents and objects that they work with, observing them directly. We don't have a chapter dedicated to this, but aspects will be found in various places throughout the book. In particular, ethnography, a form of observation deriving from anthropology, has become very influential and is discussed in Chapter 13. We will look at some ways of addressing this stage in Section 5.4.

Analysis The results of observation and interview need to be ordered in some way to bring out key issues and communicate with later stages of design. Chapter 15 and part of Chapter 18 deal with task models, which are a means to capture how people carry out the various tasks that are part of their work and life. In this chapter (Section 5.5), we will look at scenarios, rich stories of interaction, which can be used in conjunction with a method like task analysis or on their own to record and make vivid actual interaction. These techniques can be used both to represent the situation as it is and also the desired situation.

Design Well, this is all about design, but there is a central stage when you move from what you want, to how to do it. There are numerous rules, guidelines and design principles that can be used to help with this and Chapter 7 discusses these in detail; whilst Chapter 10 looks at how to design taking into account many different kinds of user. We need to record our design choices in some way and there are various notations and methods to do this, including those used to record the existing situation. Chapters 16, 17 and 18 deal with ways of modeling and describing interaction. In this chapter, Section 5.6 will look at some simple notations for designing navigation within a system and some basic heuristics to guide the design of that navigation. Section 5.7 will look more closely at the layout of individual screens. It is at this stage also where input from theoretical work is most helpful, including cognitive models, organizational issues and understanding communication (Chapters 12, 13 and 14).

Iteration and prototyping Humans are complex and we cannot expect to get designs right first time. We therefore need to evaluate a design to see how well it is working and where there can be improvements. We will discuss some techniques for evaluation in Chapter 9. Some forms of evaluation can be done using the design on paper, but it is hard to get real feedback without trying it out. Most user interface design therefore involves some form of prototyping, producing early versions of systems to try out with real users. We'll discuss this in Section 5.8.

Implementation and deployment Finally, when we are happy with our design, we need to create it and deploy it. This will involve writing code, perhaps making hardware, writing documentation and manuals – everything that goes into a real system that can be given to others. Chapter 8 will deal with software architectures for user interfaces and there are details about implementing groupware in Chapter 19 and web interfaces in Chapter 21.

If you read all the chapters and look at all the techniques you might think 'help! how can I ever do all this?'. Of course the answer is you can't. Your time is limited – there is a trade-off between the length of the design period and the quality of the final design. This means one sometimes has to accept a design as final even if it is not perfect: it is often better to have a product that is acceptable but on time and to cost than it is to have one that has perfect interaction but is late and over budget.

It is easy to think that the goal, especially of the iterative stages, is to find usability problems and fix them. As you experience real designs, however, you soon find that the real problem is not to find faults – that is easy; nor to work out how to fix them – that may not be too difficult; instead the issue is: which usability problems is it worth fixing?

In fact, if you ever come across a system that seems to be perfect it is a badly designed system – badly designed not because the design is bad, but because too much effort will have been spent in the design process itself. Just as with all trade-offs, it may be possible to find radically different solutions that have a major effect but are cheap to implement. However, it is best not to plan assuming such bolts of inspiration will strike when wanted!

5.4 USER FOCUS

As we've already said, the start of any interaction design exercise must be the intended user or users. This is often stated as:

know your users

Because this sounds somewhat like a commandment it is sometimes even written 'know thy user' (and originally 'know the user' [162]). Note, too, a little indecision about user/users – much of traditional user interface design has focussed on a single user. We will discuss issues of collaboration extensively in Chapters 13 and 19, but even at this stage it is important to be aware that there is rarely one user of a system. This doesn't mean that every system is explicitly supporting collaboration like email does. However, almost every system has an impact beyond the person immediately using it.

Think about a stock control system. The warehouse manager queries the system to find out how many six-inch nails are in stock – just a single user? Why did he do this? Perhaps a salesperson has been asked to deliver 100,000 six-inch nails within a fortnight and wants to know if the company is able to fulfill the order in time. So the act of looking at the stock control system involves the warehouse manager, the salesperson and the client. The auditors want to produce a valuation of company assets including stock in hand, the assistant warehouse manager needs to update the stock levels while his boss is on holiday.

Over time many people are affected directly or indirectly by a system and these people are called *stakeholders* (see also Chapter 13). Obviously, tracing the tenuous links between people could go on for ever and you need to draw boundaries as to whom you should consider. This depends very much on the nature of the systems being designed, but largely requires plain common sense.

So, how do you get to know your users?

■ Who are they?

Of course, the first thing to find out is who your users are. Are they young or old, experienced computer users or novices? As we saw with the stock control system, it may not be obvious who the users are, so you may need to ask this question again as you find out more about the system and its context. This question becomes harder to answer if you are designing generic software, such as a word processor, as there are many different users with different purposes and characteristics. A similar problem arises with many websites where the potential visitors are far from homogenous. It may be tempting to try to think of a generic user with generic skills and generic goals; however, it is probably better, either instead or in addition, to think of several specific users.

■ Probably *not* like you!

When designing a system it is easy to design it as if *you* were the main user: you assume your own interests and abilities. So often you hear a designer say 'but it's obvious what to do'. It may be obvious for her! This is not helped by the fact that many software houses are primarily filled with male developers. Although individuals differ a lot there is a tendency for women to have better empathetic skills.

■ Talk to them.

It is hard to get yourself inside someone else's head, so the best thing is usually to ask them. This can take many forms: structured interviews about their job or life, open-ended discussions, or bringing the potential users fully into the design process. The last of these is called *participatory design* (see Chapter 13, Section 13.3.4). By involving users throughout the design process it is possible to get a deep knowledge of their work context and needs. The obvious effect of this is that it produces better designs. However, there is a second motivational effect, perhaps at least as important as the quality of the design. By being involved, users come to 'own' the design and become champions for it once deployed. Recall that a system must be not only useful and usable, but also *used*.

People may also be able to tell you about how things *really* happen, not just how the organization says they *should* happen. To encourage users to tell you this, you will need to win their trust, since often the actual practices run counter to corporate policy. However it is typically these ad hoc methods that make organizations work, not the official story!

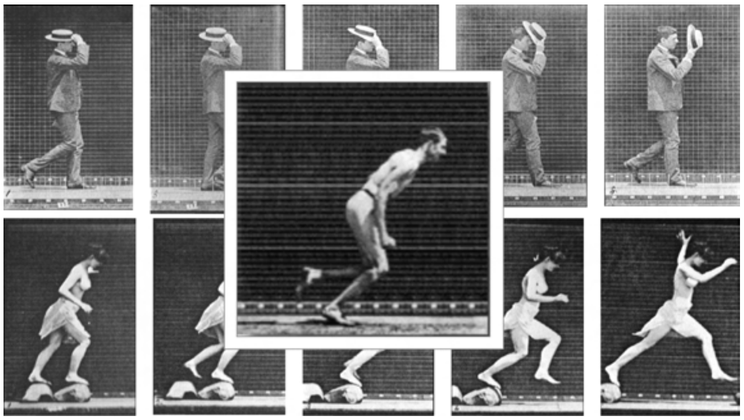


Figure 5.2 Eadweard Muybridge's time-lapse photography. Source for top plate and middle plate: Kingston Museum and Heritage Service; source for bottom plate: V&A Images, The Victoria and Albert Museum, London

- Watch them.

Although what people tell you is of the utmost importance, it is not the whole story.

When black-belt judo players are asked how they throw an opponent, their explanations do not match what they actually do. Think about walking – how do your legs and arms move? It is harder than you would think! Although people have run since the earliest times, it was only with Eadweard Muybridge's pioneering time-lapse photography in the 1870s that the way people actually walk, run and move became clear (see Figure 5.2). This is even more problematic with intellectual activities as it is notoriously difficult to introspect.

A professional in any field is very practiced and can *do* things in the domain. An academic in the same field may not be able to do things, but she knows *about* the things in the domain. These are different kinds of knowledge and skill. Sometimes people know both, but not necessarily so. The best sports trainers may not be the best athletes, the best painters may not be the best art critics.

Because of this it is important to watch what people do as well as hear what they say. This may involve sitting and taking notes of how they spend a day, watching particular activities, using a video camera or tape recorder. It can be done in an informal manner or using developed methods such as ethnography or contextual inquiry, which we will discuss in Chapter 13.

Sometimes users can be involved in this; for example, asking them to keep a diary or having a 15-minute buzzer and asking them to write down what they are doing when the buzzer sounds. Although this sounds just like asking the users what they do, the structured format helps them give a more accurate answer.

Another way to find out what people are doing is to look at the artifacts they are using and creating. Look at a typical desk in an office. There are papers, letters, files, perhaps a stapler, a computer, sticky notes . . . Some of these carry information, but if they were only important for the information in them they could equally well be

Design Focus

Cultural probes

Traditional ethnography has involved watching people and being present. There is always a disruptive effect when someone is watching, but in, say, an office, after a while the ethnographer becomes 'part of the wallpaper' and most activities carry on as normal. However, in some environments, for example the home or with psychiatric patients, it is hard to go and watch people for long periods if at all. Cultural probes have been used as one way to gather rich views of an area without intrusion. These were originally developed as prompts for design [146], but have also been adopted as an added method for ethnography [170].

Source: Photograph courtesy of William W. Gaver

Cultural probes are small packs of items designed to provoke and record comments in various ways. They are given to people to take away and to open and use in their own environment. For example, one probe pack for the domestic environment includes a glass with a paper sleeve. You use the glass to listen to things and then write down what you hear. The same probe pack contains a repackaged disposable camera and a small solid-state voice recorder. When the packs are returned, the notes, recordings, photos, etc., are used as a means of understanding what is significant and important for the people in the environment and as a means of enculturing designers.

For more see /e3/online/cultural-probes/ and www.crd.rca.ac.uk/equator/domestic_probes.html

in the filing cabinet and just taken out when needed. The sticky note on the edge of Brian's screen saying 'book table' is not just information that he needs to book a restaurant table. The fact that it is on his screen is reminding him that something needs to be done. In Chapter 18 we will look at the role of artifacts in detail.

Betty is 37 years old. She has been Warehouse Manager for five years and has worked for Simpkins Brothers Engineering for 12 years. She didn't go to university, but has studied in her evenings for a business diploma. She has two children aged 15 and 7 and does not like to work late. She did part of an introductory in-house computer course some years ago, but it was interrupted when she was promoted and could no longer afford to take the time. Her vision is perfect, but her right-hand movement is slightly restricted following an industrial accident three years ago. She is enthusiastic about her work and is happy to delegate responsibility and take suggestions from her staff. However, she does feel threatened by the introduction of yet another new computer system (the third in her time at SBE).

Figure 5.3 Persona – a rich description of Betty the Warehouse Manager

In all these observational methods one should not just stop at the observation, but go back and discuss the observations with the users. Even if they were not previously aware of what they were doing, they are likely to be able to explain when shown. The observations tell you *what* they do, they will tell you *why*.

- Use your imagination.

Even if you would like to involve many users throughout your design exercise this will not always be possible. It may be too costly, it may be hard to get time with them (e.g. hospital consultant), it may be that there are just too many (e.g. the web). However, even if you cannot involve actual users you can at least try to imagine their experiences.

Now this is very dangerous! It would be easy to think, 'if I were a warehouse manager I would do this'. The issue is not what *you* would do in the user's shoes but what *they* would do. This requires almost a kind of method acting. Imagine being a warehouse manager. What does the word 'undo' in the menu mean to him?

One method that has been quite successful in helping design teams produce user-focussed designs is the *persona*. A persona is a rich picture of an imaginary person who represents your core user group. Figure 5.3 gives an example persona of Betty the warehouse manager. A design team will have several of these personae covering different types of intended users and different roles. The personae will themselves be based on studies of actual users, observation, etc. When a design solution is proposed the team can ask, 'how would Betty react to this?'. The detail is deliberately more than is strictly necessary, but this is essential. It is only by feeling that Betty is a real person that the team can start to imagine how she will behave.

5.5 SCENARIOS

Scenarios are stories for design: rich stories of interaction. They are perhaps the simplest design representation, but one of the most flexible and powerful. Some scenarios are quite short: 'the user intends to press the "save" button, but accidentally

Brian would like to see the new film *Moments of Significance* and wants to invite Alison, but he knows she doesn't like 'arty' films. He decides to take a look at it to see if she would like it and so connects to one of the movie-sharing networks. He uses his work machine as it has a higher bandwidth connection, but feels a bit guilty. He knows he will be getting an illegal copy of the film, but decides it is OK as he is intending to go to the cinema to watch it. After it downloads to his machine he takes out his new personal movie player. He presses the 'menu' button and on the small LCD screen he scrolls using the arrow keys to 'bluetooth connect' and presses the 'select' button. On his computer the movie download program now has an icon showing that it has recognized a compatible device and he drags the icon of the film over the icon for the player. On the player the LCD screen says 'downloading now', with a per cent done indicator and small whirling icon.

During lunchtime Brian takes out his movie player, plugs in his earphones and starts to watch. He uses the arrow keys to skip between portions of the film and decides that, yes, Alison would like it. Then he feels a tap on his shoulder. He turns round. It is Alison. He had been so absorbed he hadn't noticed her. 'What are you watching', she says. 'Here, listen', he says and flicks a small switch. The built-in directional speaker is loud enough for both Brian and Alison to hear, but not loud enough to disturb other people in the canteen. Alison recognizes the film from trailers, 'surprised this is out yet' she says. 'Well actually . . .', Brian confesses, 'you'd better come with me to see it and make an honest man of me'. 'I'll think about it', she replies.

Figure 5.4 Scenario for proposed movie player

presses the "quit" button so loses his work'. Others are focussed more on describing the situation or context.

Figure 5.4 gives an example of a scenario for the personal movie player. Like the persona it is perhaps more detailed than appears necessary, but the detail helps make the events seem real. The figure shows plain text, but scenarios can be augmented by sketches, simulated screen shots, etc. These sketches and pictures are called *storyboards* and are similar to the techniques used in film making to envisage plot-lines.

Where the design includes physical artifacts the scenarios can be used as a script to act out potential patterns of use. For example, we might imagine a digital Swiss army knife, which has a small LCD screen and uses the toothpick as a stylus. The knife connects to the internet via a wireless link through your phone and gives interesting tips from other Swiss army knife users. Try getting two together at a party – you will see this would appeal! It sounds like a great design idea – but wait, try acting out the use. If you have a Swiss army knife, use it, or use something penknife-sized if you don't. The tip on the LCD says, 'open the stone remover': a small LED glows near the right blade – you open it. 'Now push the blade into the rubber of the grommet', it says. You do this and then look for the next instruction. Look at the knife in your hand . . . oops, your thumb is covering where the screen would be. Perhaps a voice interface would be better.

You can see already how scenarios force you to think about the design in detail and notice potential problems before they happen. If you add more detail you can get to a blow-by-blow account of the user–system interactions and then ask 'what is the user intending now?'; 'what is the system doing now?'. This can help to verify that the design would make sense to the user and also that proposed implementation architectures would work.

In addition scenarios can be used to:

Communicate with others – other designers, clients or users. It is easy to misunderstand each other whilst discussing abstract ideas. Concrete examples of use are far easier to share.

Validate other models A detailed scenario can be 'played' against various more formal representations such as task models (discussed in Chapter 15) or dialog and navigation models (Chapter 16 and below).

Express dynamics Individual screen shots and pictures give you a sense of what a system would look like, but not how it behaves.

In the next section we will discuss ways of describing the patterns of interaction with a system. These are more complex and involve networks or hierarchies. In contrast scenarios are linear – they represent a single path amongst all the potential interactions.

This linearity has both positive and negative points:

Time is linear Our lives are linear as we live in time and so we find it easier to understand simple linear narratives. We are natural storytellers and story listeners.

But no alternatives Real interactions have choices, some made by people, some by systems. A simple scenario does not show these alternative paths. In particular, it is easy to miss the unintended things a person may do.

Scenarios are a resource that can be used and reused throughout the design process: helping us see what is wanted, suggesting how users will deal with the potential design, checking that proposed implementations will work, and generating test cases for final evaluation.

For more examples of scenarios see: /e3/online/scenario/

5.6 NAVIGATION DESIGN

As we stressed, the object of design is not just a computer system or device, but the socio-technical intervention as a whole. However, as design progresses we come to a point where we do need to consider these most tangible outputs of design.

Imagine yourself using a word processor. You will be doing this in some particular social and physical setting, for a purpose. But now we are focussing on the computer system itself. You interact at several levels:

Widgets The appropriate choice of widgets and wording in menus and buttons will help you know how to use them for a particular selection or action.

Screens or windows You need to find things on the screen, understand the logical grouping of buttons.

Table 5.1 Levels of interaction

PC application	Website	Physical device
Widgets	Form elements, tags and links	Buttons, dials, lights, displays
Screen design	Page design	Physical layout
Navigation design	Site structure	Main modes of device
Other apps and operating system	The web, browser, external links	The real world!

Navigation within the application You need to be able to understand what will happen when a button is pressed, to understand where you are in the interaction.

Environment The word processor has to read documents from disk, perhaps some are on remote networks. You swap between applications, perhaps cut and paste.

You can see similar levels in other types of application and device, as Table 5.1 shows. There are differences; for example, in the web we have less control of how people enter a site and on a physical device we have the same layout of buttons and displays no matter what the internal state (although we may treat them differently).

We discussed graphical user interface widgets in Chapter 3 and in the next section we will look at details of screen design. In this section we will look mainly at navigation design, that is the main screens or modes within a system and how they interconnect. We will also briefly consider how this interacts with the wider environment.

Just in case you haven't already got the idea, the place to start when considering the structure of an application is to think about actual use:

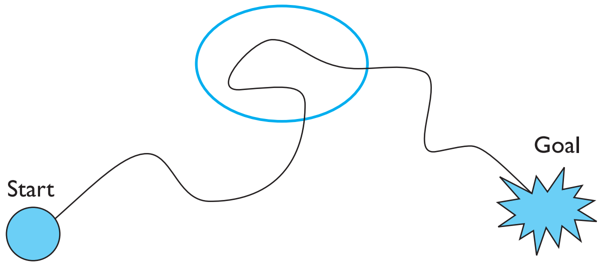
- who is going to use the application?
- how do they think about it?
- what will they do with it?

This can then drive the second task – thinking about structure. Individual screens or the layout of devices will have their own structure, but this is for the next section. Here we will consider two main kinds of issue:

- local structure
  - looking from one screen or page out
- global structure
  - structure of site, movement between screens.

5.6.1 Local structure

Much of interaction involves goal-seeking behavior. Users have some idea of what they are after and a partial model of the system. In an ideal world if users had perfect knowledge of what they wanted and how the system worked they could simply take the shortest path to what they want, pressing all the right buttons and links. However, in a world of partial knowledge users meander through the system. The important thing is not so much that they take the most efficient route, but that at each point in the interaction they can make some assessment of whether they are getting closer to their (often partially formed) goal.



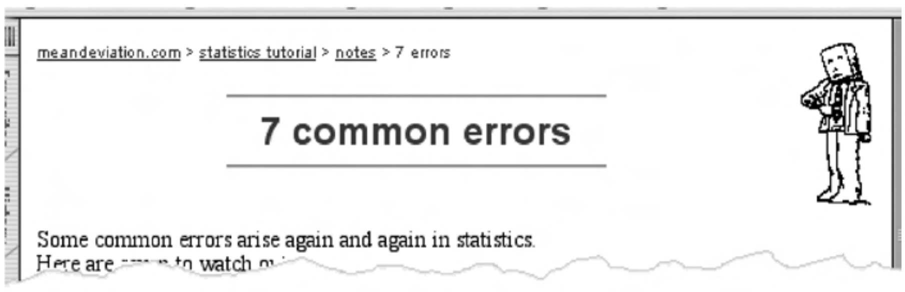
To do this goal seeking, each state of the system or each screen needs to give the user enough knowledge of what to do to get closer to their goal. In Chapter 7 we will look at various design rules, some of which address this issue. To get you started, here are four things to look for when looking at a single web page, screen or state of a device.

- knowing where you are
- knowing what you can do
- knowing where you are going – or what will happen
- knowing where you've been – or what you've done.

The screen, web page or device displays should make clear *where you are* in terms of the interaction or state of the system. Some websites show 'bread crumbs' at the top of the screen, the path of titles showing where the page is in the site (Figure 5.5). Similarly, in the scenario in Figure 5.4, the personal movie player says 'downloading now', so Brian knows that it is in the middle of downloading a movie from the PC.

It is also important to know *what you can do* – what can be pressed or clicked to go somewhere or do something. Some web pages are particularly bad in that it is unclear which images are pure decoration and which are links to take you somewhere.

On the web the standard underlined links make it clear which text is clickable and which is not. However, in order to improve the appearance of the page many sites change the color of links and may remove the underline too. This is especially confusing if underline is then used as simple emphasis on words that are not links! The


meandeviation.com > statistics tutorial > notes > 7 errors

7 common errors

Some common errors arise again and again in statistics.
Here are

Figure 5.5 Breadcrumbs. Screen shot frame reprinted by permission from Microsoft Corporation

trade-off between appearance and ease of use may mean that this is the right thing to do, but you should take care before confusing the user needlessly.

Chic design is also a problem in physical devices. One of the authors was once in a really high-class hotel and found he could not locate the flush in the toilet. Only after much fumbling did he discover that one of the tiles could be pressed. The 'active' tile was level with the rest of the tiled wall – a very clean design, but not very usable!

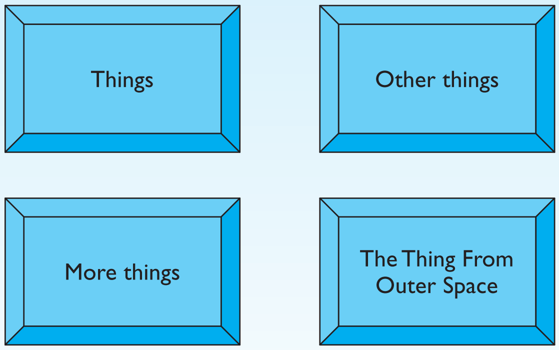
You then need to know *where you are going* when you click a button or *what will happen*. Of course you can try clicking the button to see. In the case of a website or information system this may mean you then have to use some sort of 'back' mechanism to return, but that is all; however, in an application or device the action of clicking the button may already have caused some effect. If the system has an easy means to undo or reverse actions this is not so bad, but it is better if users do not have to use this 'try it and see' interaction. Where response times are slow this is particularly annoying.

Remember too that icons are typically not self-explanatory and should always be accompanied by labels or at the very least tooltips or some similar technique. A picture paints a thousand words, but typically only when explained first using fifteen hundred!

DESIGN FOCUS

Beware the big button trap

Public information systems often have touchscreens and so have large buttons. Watch someone using one of these and see how often they go to the wrong screen and have to use 'back' or 'home' to try again. If you look more closely you will find that each button has only one or two words on it giving the title of the next screen, and possibly some sort of icon. Quite rightly, the button label will be in a large font as users may have poor eyesight.



It is hard to choose appropriate labels that mean the same for everyone, especially when the breadth of the screen hierarchy is fixed by the maximum number of buttons. So it is no wonder that people get confused. However, there is usually plenty of room for additional explanation in a smaller font, possibly just the next level of button labels, or a sentence of explanation. It may not look as pretty, but it may mean that people actually find the information they are looking for.

Special care has to be taken if the same command or button press means something different in different contexts. These different contexts that change the interpretation of commands are called *modes*. Many older text editors would interpret pressing 'x' to mean 'enter me into the text' in a normal typing mode, but 'exit' in a special command mode. If modes are clearly visible or audible this is less of a problem and in Chapter 3 (Section 3.6.7) we saw how palettes are one way to achieve this. In general, modes are less of a problem in windowed systems where the mode is made apparent by the current window (if you remember which it is). However, physical devices may have minimal displays and may be operated without visual attention.

Finally, if you have just done some major action you also want some sort of confirmation of *what you've done*. If you are faultless and have perfect knowledge, of course you will be sure that you have hit the right key and know exactly what

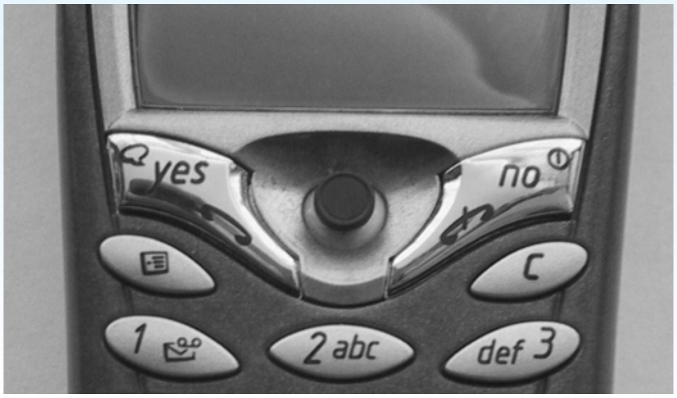
DESIGN FOCUS

Modes

Alan's mobile phone has a lock feature to prevent accidental use. To remove the lock he has to press the 'C' (cancel) button which then asks for an additional 'yes' to confirm removing the lock. So, in 'locked' mode, 'C' followed by 'yes' means 'turn off lock' and these are the most frequent actions when Alan takes the phone from his pocket.

However, Alan is forgetful and sometimes puts the phone in his pocket unlocked. This leads to occasional embarrassing phone calls and also to another problem.

The 'yes' button is quite big and so this is often pressed while in his pocket. This puts the phone into 'dial recent numbers' mode with a list of recent calls on screen. In this mode, pressing 'C' gives a prompt 'delete number' and pressing 'yes' then deletes the number from the phone's address book. Unhappily, this often means he takes the phone from his pocket, automatically presses 'C', 'yes' only to see as he looks down to the handset the fatal words 'number deleted'. Of course there is no undo!



will happen. Remember, too, that to know what will happen, you would need to know everything about the internal state of the system and things outside, like the contents of files, networked devices, etc., that could affect it. In other words, if you were omniscient you could do it. For lesser mortals the system needs to give some *feedback* to say what has happened.

In an information system, there is a related but slightly different issue, which is to know *where you have been*. This helps you to feel in control and understand your navigation of the information space. The feeling of disorientation when you do not have sufficient means to know where you are and where you have been has been called 'lost in hyperspace'. Most web browsers offer a history system and also a 'back' button that keeps a list of recently visited pages.

5.6.2 Global structure – hierarchical organization

We will now look at the overall structure of an application. This is the way the various screens, pages or device states link to one another.

One way to organize a system is in some form of hierarchy. This is typically organized along functional boundaries (that is, different kinds of things), but may be organized by roles, user type, or some more esoteric breakdown such as modules in an educational system.

The hierarchy links screens, pages or states in logical groupings. For example, Figure 5.6 gives a high-level breakdown of some sort of messaging system. This sort of hierarchy can be used purely to help during design, but can also be used to structure the actual system. For example, this may reflect the menu structure of a PC application or the site structure on the web.

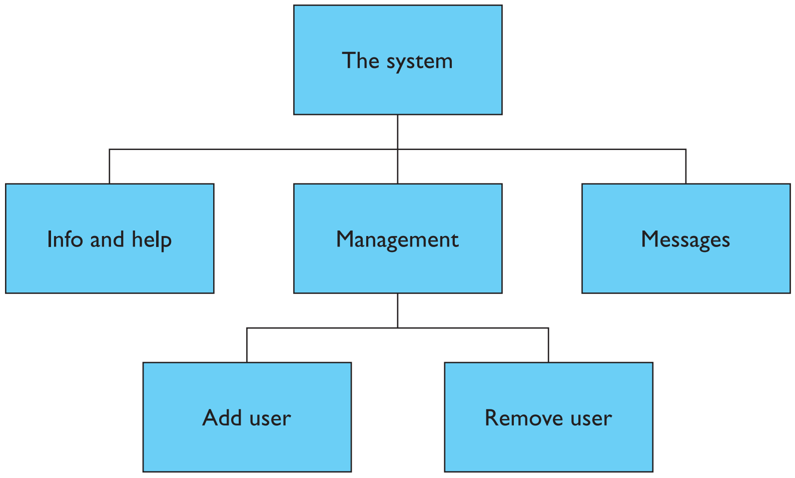


Figure 5.6 Application functional hierarchy

Any sort of information structuring is difficult, but there is evidence that people find hierarchies simpler than most. One of the difficulties with organizing information or system functionality is that different people have different internal structures for their knowledge, and may use different vocabulary. This is one of the places where a detailed knowledge of the intended users is essential: it is no good creating a hierarchy that the designers understand, but not the users . . . and all too commonly this is exactly what happens.

However much you think you have got the wording and categories right, because there are different users it is inevitable that not everyone will understand it perfectly. This is where clear guidance as suggested in Section 5.6.1 (knowing where you are going – or what will happen) is essential, as well as the means to allow users to change their mind if they make the wrong decisions.

There is also evidence that deep hierarchies are difficult to navigate, so it is better to have broad top-level categories, or to present several levels of menu on one screen or web page. Miller's magic number of 7 ± 2 for working memory capacity (see Chapter 1, Section 1.3.2) is often misused in this context. Many guidelines suggest that menu breadth, that is the number of choices available at each level in the menu, should be around seven. However, Miller's result applies only to working memory, not visual search. In fact, optimal breadth can be quite large, perhaps 60 or more items for a web index page if the items are organized in such a way that the eye can easily find the right one [206]. (See /e3/online/menu-breadth/ for more on optimal menu breadth.) Of course, to organize the items on the page requires further classification. However, here the critical thing is the naturalness of the classification, which itself may depend on the user's purpose. For example, if the user wants to look up information on a particular city, an alphabetical list of all city names would be fast, but for other purposes a list by region would be more appropriate.

5.6.3 Global structure – dialog

In a pure information system or static website it may be sufficient to have a fully hierarchical structure, perhaps with next/previous links between items in the same group. However, for any system that involves doing things, constantly drilling down from one part of the hierarchy to another is very frustrating. Usually there are ways of getting more quickly from place to place. For example, in a stock control system there may be a way of going from a stock item to all orders outstanding on that item and then from an order to the purchase record for the customer who placed the order. These would each be in a very different part of a hierarchical view of the application, yet directly accessible from one another.

As well as these cross-links in hierarchies, when you get down to detailed interactions, such as editing or deleting a record, there is obviously a flow of screens and commands that is not about hierarchy. In HCI the word 'dialog' is used to refer to this pattern of interactions between the user and a system.

Consider the following fragment from a marriage service:

Minister:	Do you *name* take this woman . . .
Man:	I do
Minister:	Do you *name* take this man . . .
Woman:	I do
Minister:	I now pronounce you man and wife

Notice this describes the general flow of the service, but has blanks for the names of the bride and groom. So it gives the pattern of the interaction between the parties, but is instantiated differently for each service. Human–computer dialog is just the same; there are overall patterns of movement between main states of a device or windows in a PC application, but the details differ each time it is run.

Recall that scenarios gave just one path through the system. To describe a full system we need to take into account different paths through a system and loops where the system returns to the same screen. There are various ways to do this, and in Chapter 16 we will expand on the wedding example and look at several different types of dialog model.

A simple way is to use a network diagram showing the principal states or screens linked together with arrows. This can:

- show what leads to what
- show what happens when
- include branches and loops
- be more task oriented than a hierarchy.

Figure 5.7 shows a network diagram illustrating the main screens for adding or deleting a user from the messaging system in Figure 5.6. The arrows show the general flow between the states. We can see that from the main screen we can get to either the 'remove user' screen or the 'add user' screen. This is presumably by selecting buttons or links, but the way these are shown we leave to detailed screen design. We can also see that from the 'add user' screen the system always returns to the main screen, but after the 'remove user' screen there is a further confirmation screen.

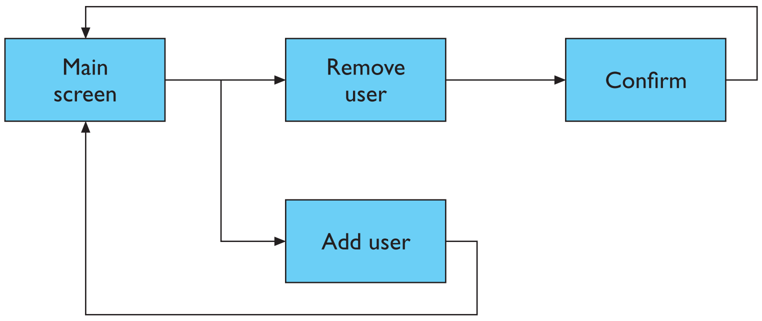


Figure 5.7 Network of screens/states

5.6.4 Wider still

Donne said ‘No man is an Iland, intire of it selfe’. This is also true of the things we design. Each sits amongst other devices and applications and this in turn has to be reflected within our design.

This has several implications:

Style issues We should normally conform to platform standards, such as positions for menus on a PC application, to ensure consistency between applications. For example, on our proposed personal movie player we should make use of standard fast-forward, play and pause icons.

Functional issues On a PC application we need to be able to interact with files, read standard formats and be able to handle cut and paste.

Navigation issues We may need to support linkages between applications, for example allowing the embedding of data from one application in another, or, in a mail system, being able to double click an attachment icon and have the right application launched for the attachment.

On the web we have the added difficulty that other sites and applications may include links that bypass our ‘home page’ and other pages and go direct into the heart of our site or web application. Also, when we link to other sites, we have no control over them or the way their content may change over time.

5.7 SCREEN DESIGN AND LAYOUT

We have talked about the different elements that make up interactive applications, but not about how we put them together. A single screen image often has to present information clearly and also act as the locus for interacting with the system. This is a complex area, involving some of the psychological understanding from Chapter 1 as well as aspects of graphical design.

The basic principles at the screen level reflect those in other areas of interaction design:

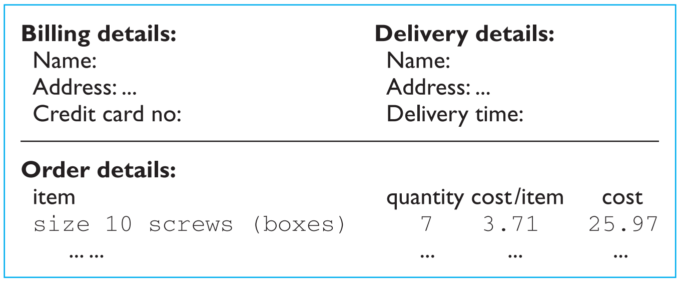
Ask What is the user doing?

Think What information is required? What comparisons may the user need to make? In what order are things likely to be needed?

Design Form follows function: let the required interactions drive the layout.

5.7.1 Tools for layout

We have a number of visual tools available to help us suggest to the user appropriate ways to read and interact with a screen or device.



```
Billing details:                        Delivery details:
  Name:                                   Name:
  Address: ...                            Address: ...
  Credit card no:                         Delivery time:
--------------------------------------------------------------------
Order details:
  item                                    quantity cost/item   cost
  size 10 screws (boxes)                     7       3.71     25.97
     ... ...                                 ...     ...       ...
```

Figure 5.8 Grouping related items in an order screen

Grouping and structure

If things logically belong together, then we should normally physically group them together. This may involve multiple levels of structure. For example, in Figure 5.8 we can see a potential design for an ordering screen. Notice how the details for billing and delivery are grouped together spatially; also note how they are separated from the list of items actually ordered by a line as well as spatially. This reflects the following logical structure:

```
Order:
    Administrative information
        Billing details
        Delivery details
    Order information
        Order line 1
        Order line 2
        ...
```

Order of groups and items

If we look at Figure 5.8 again we can see that the screen seems to naturally suggest reading or filling in the billing details first, followed by the delivery details, followed by the individual order items. Is this the right order?

In general we need to think: what is the natural order for the user? This should normally match the order on screen. For data entry forms or dialog boxes we should also set up the order in which the tab key moves between fields.

Occasionally we may also want to force a particular order; for example we may want to be sure that we do not forget the credit card details!

Decoration

Again looking at Figure 5.8, we can see how the design uses boxes and a separating line to make the grouping clear. Other decorative features like font style, and text or background colors can be used to emphasize groupings. Look at the microwave control

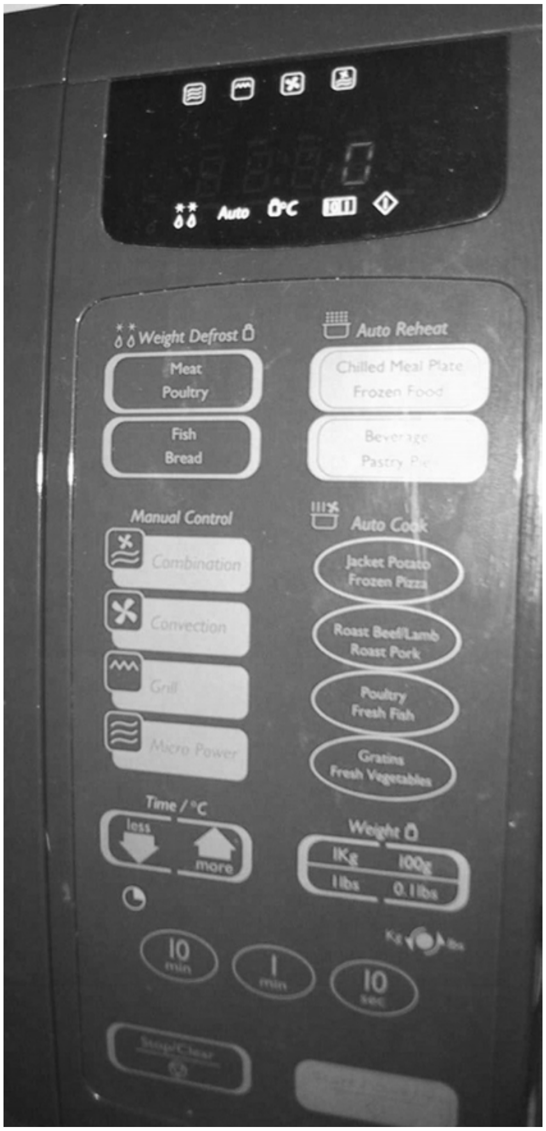


Figure 5.9 Microwave control panel

panel in Figure 5.9. See how the buttons differ in using the foreground and background colors (green and gold) so that groups are associated with one another. See also how the buttons are laid out to separate them into groups of similar function.

Alignment

Alignment of lists is also very important. For users who read text from left to right, lists of text items should normally be aligned to the left. Numbers, however, should

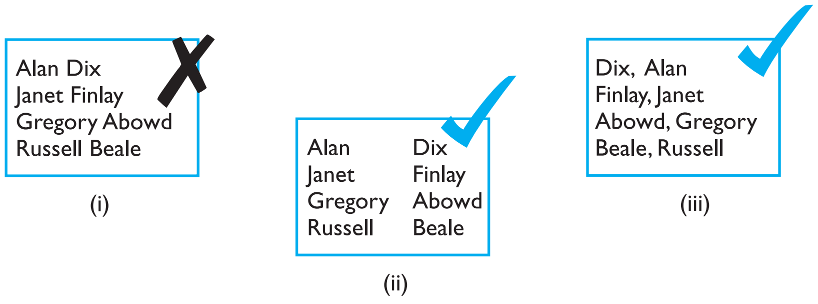


Figure 5.10 Looking up surnames

normally be aligned to the right (for integers) or at the decimal point. This is because the shape of the column then gives an indication of magnitude – a sort of mini-histogram. Items like names are particularly difficult. Consider list (i) in Figure 5.10. It is clearly hard to look someone up if you only know their surname. To make it easy, such lists should be laid out in columns as in (ii), or have forename and surname reversed as in (iii). (The dates in Figure 5.13, Section 5.7.3, pose similar problems, as the years do not align, even when the folder is sorted by date.)

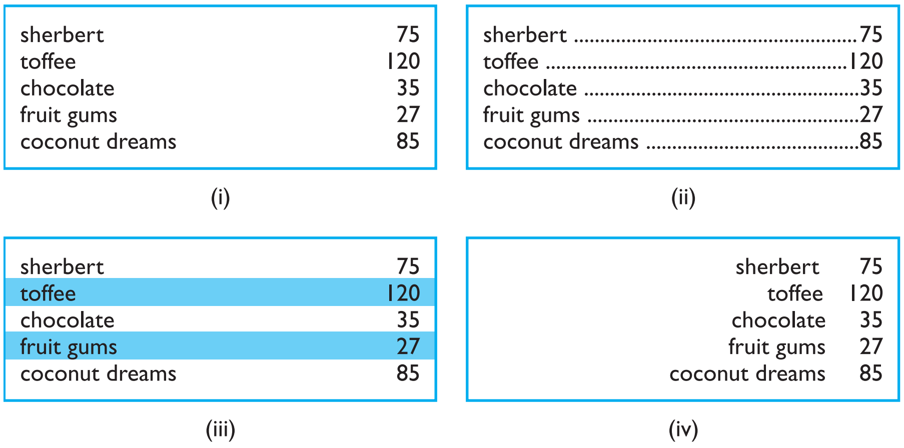
Design Focus

Alignment and layout matter

Look quickly at these two columns of numbers and try to find the biggest number in each column.

532.56	627.865
179.3	1.005763
256.317	382.583
15	2502.56
73.948	432.935
1035	2.0175
3.142	652.87
497.6256	56.34

Multiple column lists require more care. Text columns have to be wide enough for the largest item, which means you can get large gaps between columns. Figure 5.11 shows an example of this (i), and you can see how hard this makes it for your eye to scan across the rows. There are several visual ways to deal with this including: (ii) 'leaders' – lines of dots linking the columns; and (iii) using soft tone grays or colors behind rows or columns. This is also a time when it may be worth breaking other



sherbert	75
toffee	120
chocolate	35
fruit gums	27
coconut dreams	85

(i)

sherbert	75
toffee	120
chocolate	35
fruit gums	27
coconut dreams	85

(ii)

sherbert	75
toffee	120
chocolate	35
fruit gums	27
coconut dreams	85

(iii)

sherbert	75
toffee	120
chocolate	35
fruit gums	27
coconut dreams	85

(iv)

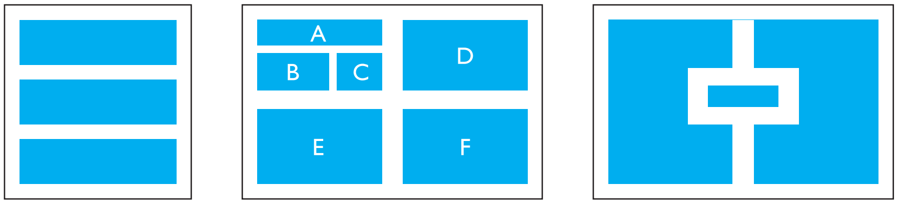
Figure 5.11 Managing multiple columns

alignment rules, perhaps right aligning some text items as in (iv). This last alternative might be a good solution if you were frequently scanning the numbers and only occasionally scanning the names of items, but not if you needed frequently to look up names (which anyway are not sorted in this figure!). You can also see that this is an example of a design trade-off – good alignment for individual columns versus ability to see relationship across rows.

White space

In typography the space between the letters is called the counter. In painting this is also important and artists may focus as much on the space between the foreground elements such as figures and buildings as on the elements themselves. Often the shape of the counter is the most important part of the composition of a painting and in calligraphy and typography the balance of a word is determined by giving an even weight to the counters. If one ignores the 'content' of a screen and instead concentrates on the counter – the space between the elements – one can get an overall feel for the layout. If elements that are supposed to be related look separate when you focus on the counter, then something is wrong. Screwing up your eyes so that the screen becomes slightly blurred is another good technique for taking your attention away from the content and looking instead at the broad structure.

Space can be used in several ways. Some of these are shown in Figure 5.12. The colored areas represent continuous areas of text or graphics. In (i) we can see space used to separate blocks as you often see in gaps between paragraphs or space between sections in a report. Space can also be used to create more complex structures. In (ii) there are clearly four main areas: ABC, D, E and F. Within one of these are three further areas, A, B and C, which themselves are grouped as A on its own, followed by B and C together. In Figure 5.12 (iii), we can see space used to highlight. This is a technique used frequently in magazines to highlight a quote or graphic.



(i) Space to separate (ii) Space to structure (iii) Space to highlight

Figure 5.12 Using white space in layout

5.7.2 User action and control

Entering information

Some of the most complicated and difficult screen layouts are found in forms-based interfaces and dialog boxes. In each case the screen consists not only of information presented to the user, but also of places for the user to enter information or select options. Actually, many of the same layout issues for data presentation also apply to fields for data entry. Alignment is still important. It is especially common to see the text entry boxes aligned in a jagged fashion because the field names are of different lengths. This is an occasion where right-justified text for the field labels may be best or, alternatively, in a graphical interface a smaller font can be used for field labels and the labels placed just above and to the left of the field they refer to.

For both presenting and entering information a clear logical layout is important. The task analysis techniques in Chapter 15 can help in determining how to group screen items and also the order in which users are likely to want to read them or fill them in. Knowing also that users are likely to read from left to right and top to bottom (depending on their native language!) means that a screen can be designed so that users encounter items in an appropriate order for the task at hand.

Knowing what to do

Some elements of a screen are passive, simply giving you information; others are active, expecting you to fill them in, or do something to them. It is often not even clear which elements are active, let alone what the effect is likely to be when you interact with them!

This is one of the reasons for platform and company style guides. If everyone designs buttons to look the same and menus to look the same, then users will be able to recognize them when they see them. However, this is not sufficient in itself. It is important that the labels and icons on menus are also clear. Again, standards can help for common actions such as save, delete or print. For more system-specific actions, one needs to follow broader principles. For example, a button says ‘bold’: does this represent the current *state* of a system or the *action* that will be performed if the button is pressed?

Affordances

These are especially difficult problems in multimedia applications where one may deliberately adopt a non-standard and avant-garde style. How are users supposed to know where to click? The psychological idea of *affordance* says that things may suggest by their shape and other attributes what you can do to them: a handle affords pulling or lifting; a button affords pushing. These affordances can be used when designing novel interaction elements. One can either mimic real-world objects directly, or try to emulate the critical aspects of those objects. What you must not do is depict a real-world object in a context where its normal affordances do not work!

Note also that affordances are not intrinsic, but depend on the background and culture of users. Most computer-literate users will click on an icon. This is not because they go around pushing pictures in art galleries, but because they have learned that this is an affordance of such objects in a computer domain. Similarly, such experienced users may well double click if a single click has no effect, yet novices would not even think of double clicking – after all, double clicking on most real buttons turns them off again!

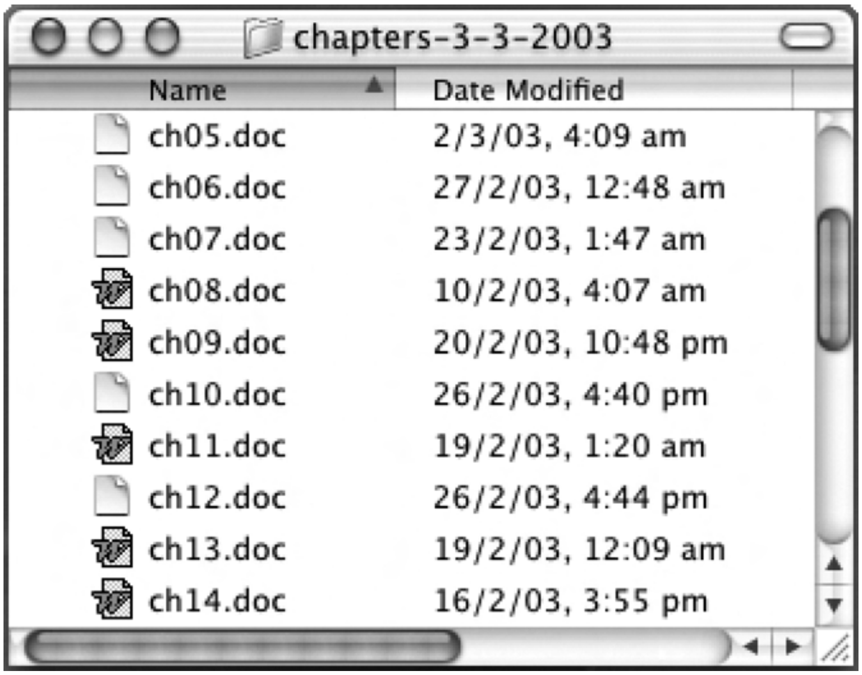
5.7.3 Appropriate appearance

Presenting information

The way of presenting information on screen depends on the kind of information: text, numbers, maps, tables; on the technology available to present it: character display, line drawing, graphics, virtual reality; and, most important of all, on the purpose for which it is being used. Consider the window in Figure 5.13. The file listing is alphabetic, which is fine if we want to look up the details of a particular file, but makes it very difficult to find recently updated files. Of course, if the list were ordered by date then it would be difficult to find a particular file. Different purposes require different representations. For more complex numerical data, we may be considering scatter graphs, histograms or 3D surfaces; for hierarchical structures, we may consider outlines or organization diagrams. But, no matter how complex the data, the principle of matching presentation to purpose remains.

The issue of presentation has been around for many years, long before computers, interactive systems or HCI! Probably the best source for this issue is Tufte's book [351]. It is targeted principally at static presentations of information, as in books, but most design principles transfer directly.

We have an advantage when presenting information in an interactive system in that it is easy to allow the user to choose among several representations, thus making it possible to achieve different goals. For example, with Macintosh folder windows (as in Figure 5.13) the user can click on a column heading and the file list is reordered, so one can look at the files by, say, name or date. This is not an excuse for ignoring the user's purpose, but means that we can plan for a range of possible uses.



Name	Date Modified
ch05.doc	2/3/03, 4:09 am
ch06.doc	27/2/03, 12:48 am
ch07.doc	23/2/03, 1:47 am
ch08.doc	10/2/03, 4:07 am
ch09.doc	20/2/03, 10:48 pm
ch10.doc	26/2/03, 4:40 pm
ch11.doc	19/2/03, 1:20 am
ch12.doc	26/2/03, 4:44 pm
ch13.doc	19/2/03, 12:09 am
ch14.doc	16/2/03, 3:55 pm

Figure 5.13 Alphabetic file listing. Screen shot reprinted by permission from Apple Computer, Inc.

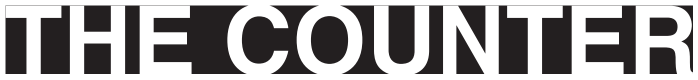
Aesthetics and utility

Remember that a pretty interface is not necessarily a good interface. Ideally, as with any well-designed item, an interface should be aesthetically pleasing. Indeed, good graphic design and attractive displays can increase users' satisfaction and thus improve productivity.

However, beauty and utility may sometimes be at odds. For example, an industrial control panel will often be built up of the individual controls of several subsystems, some designed by different teams, some bought in. The resulting inconsistency in appearance may look a mess and suggest tidying up. Certainly some of this inconsistency may cause problems. For example, there may be a mix of telephone-style and calculator-style numeric keypads. Under stress it would be easy to mis-key when swapping between these. However, the diversity of controls can also help the operator keep track of which controls refer to which subsystem – any redesign must preserve this advantage.

The conflict between aesthetics and utility can also be seen in many 'well-designed' posters and multimedia systems. In particular, the backdrop behind text must have low contrast in order to leave the text readable; this is often not the case and graphic designers may include excessively complex and strong backgrounds because they look good. The results are impressive, perhaps even award winning, but completely unusable!

On a more positive note, careful application of aesthetic concepts can also aid comprehensibility. An example of this is the idea of the counter and use of space that we discussed earlier. In consumer devices these aesthetic considerations may often be the key differentiator between products, for example, the sleek curves of a car. This is not missed by designers of electronic goods: devices are designed to be good to touch and feel as well as look at and this is certainly one of the drivers for the futuristic shapes of the Apple iMac family.



Making a mess of it: color and 3D

One of the worst features in many interfaces is their appalling use of color. This is partly because many monitors only support a limited range of primary colors and partly because, as with the overuse of different fonts in word processors, the designer got carried away. Aside from issues of good taste, an overuse of color can be distracting and, remembering from Chapter 1 that a significant proportion of the population is color blind, may mean that parts of the text are literally invisible to some users. In general, color should be used sparingly and not relied upon to give information, but rather to reinforce other attributes.

The increasing use of 3D effects in interfaces has posed a whole new set of problems for text and numerical information. Whilst excellent for presenting physical information and certain sorts of graphs, text presented in perspective can be very difficult to read and the all too common 3D pie chart is all but useless. We will discuss ways to make 3D actually useful for visualization in Chapter 20.

Design Focus

Checking screen colors

Even non-color-blind users will find it hard to read text where the intensity of the text and background are similar. A good trick is to adjust the color balance on your monitor so that it is reduced to grays, or to print screens on a black and white printer. If your screen is unreadable in grayscale then it is probably difficult to read in full color.

Localization / internationalization

If you are working in a different country, you might see a document being word processed where the text of the document and the file names are in the local language, but all the menus and instructions are still in English. The process of making software suitable for different languages and cultures is called *localization* or *internationalization*.

It is clear that words have to change and many interface construction toolkits make this easy by using *resources*. When the program uses names of menu items, error messages and other text, it does not use the text directly, but instead uses a resource identifier, usually simply a number. A simple database is constructed separately that binds these identifiers to particular words and phrases. A different resource database is constructed for each language, and so the program can be customized to use in a particular country by simply choosing the appropriate resource database.

However, changing the language is only the simplest part of internationalization. Much of the explicit guidance on alignment and layout is dependent on a left-to-right, top-to-bottom language such as English and most European languages. This obviously changes completely for other types of language. Furthermore, many icons and images are only meaningful within a restricted cultural context. Despite the apparent international hegemony of Anglo-American culture, one cannot simply assume that its symbols and norms will be universally understood. A good example of this is the use of ticks ✓ and crosses ✗. In Anglo-American culture these represent opposites, positive and negative, whereas in most of Europe the two are interchangeable.

5.8 ITERATION AND PROTOTYPING

Because human situations are complex and designers are not infallible it is likely that our first design will not be perfect! For this reason, almost all interaction design includes some form of iteration of ideas. This often starts early on with paper designs and storyboards being demonstrated to colleagues and potential users. Later in the design process one might use mockups of physical devices or tools such as Shockwave or Visual Basic to create prototype versions of software.

Any of these prototypes, whether paper-based or running software, can then be evaluated to see whether they are acceptable and where there is room for improvement. This sort of evaluation, intended to improve designs, is called *formative* evaluation. This is in contrast to *summative* evaluation, which is performed at the end to verify whether the product is good enough. Chapter 9 considers evaluation in detail. One approach is to get an expert to use a set of guidelines, for example the ‘knowing where you are’ list above, and look screen by screen to see if there are any violations. The other main approach is to involve real users either in a controlled experimental setting, or ‘in the wild’ – a real-use environment.

The result of evaluating the system will usually be a list of faults or problems and this is followed by a redesign exercise, which is then prototyped, evaluated . . . Figure 5.14 shows this process. The end point is when there are no more problems that can economically be fixed.

So iteration and prototyping are the universally accepted ‘best practice’ approach for interaction design. However, there are some major pitfalls of prototyping, rarely acknowledged in the literature.

Prototyping is an example of what is known as a *hill-climbing* approach. Imagine you are standing somewhere in the open countryside. You walk uphill and keep going uphill as steeply as possible. Eventually you will find yourself at a hill top. This

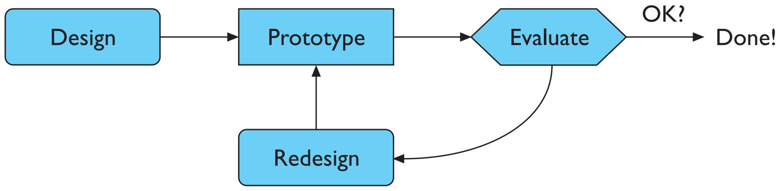


Figure 5.14 Role of prototyping

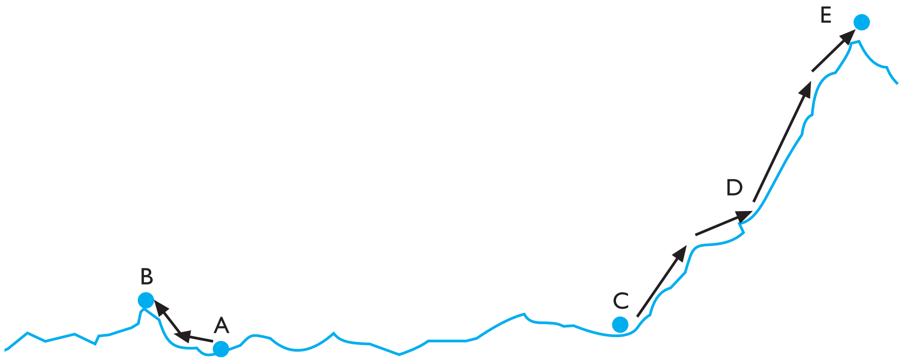


Figure 5.15 Moving little by little . . . but to where?

is exactly how iterative prototyping works: you start somewhere, evaluate it to see how to make it better, change it to make it better and then keep on doing this until it can't get any better.

However, hill climbing doesn't always work. Imagine you start somewhere near Cambridge, UK. If you keep moving uphill (and it is very difficult to work out which direction that is because it is very flat!), then eventually you would end up at the top of the Gog Magog hills, the nearest thing around . . . all of 300 feet. However, if you started somewhere else you might end up at the top of the Matterhorn. Hill-climbing methods always have the potential to leave you somewhere that is the best in the immediate area, but very poor compared with more distant places. Figure 5.15 shows this schematically: if you start at A you get trapped at the *local maximum* at B, but if you start at C you move up through D to the *global maximum* at E.

This problem of getting trapped at local maxima is also possible with interfaces. If you start with a bad design concept you may end at something that is simply a tidied up version of that bad idea!

From this we can see that there are two things you need in order for prototyping methods to work:

1. To understand what is wrong and how to improve.
2. A good start point.

The first is obvious; you cannot iterate the design unless you know what must be done to improve it. The second, however, is needed to avoid local maxima. If you wanted to climb as high as you could, you would probably book a plane to the Himalayas, not Cambridgeshire.

A really good designer might guess a good initial design based on experience and judgment. However, the complexity of interaction design problems means that this insight is hard. Another approach, very common in graphical design, is to have several initial design ideas and drop them one by one as they are developed further. This is a bit like parachuting 10 people at random points of the earth. One of them is perhaps likely to end up near a high mountain.

One of the things that theoretical methods and models, as found in Part 3 of this book, can do is to help us with both (1) and (2).

5.9 SUMMARY

We have seen that design in HCI is not just about creating devices or software, but instead is about the whole interaction between people, software and their environment. Because of this it is good to see the product of design not just as the obvious artifacts but as the whole intervention that changes the existing situation to a new one.

In Section 5.2, design was defined as 'achieving goals within constraints'. In the case of interaction design the goals are about improving some aspect of work, home or leisure using technology. The constraints remind us that the final design will inevitably involve trade-offs between different design issues and furthermore should never be 'perfect' as cost and timeliness should prevent indefinite tinkering. To achieve good design we must understand our materials and in the case of interaction design these materials include not just the computers and technical devices, but also humans. If we treated humans in design with only as much care as physical materials it is clear that 'human error' after accidents would be regarded as 'design error' – a good designer understands the natural limitations of ordinary people.

Section 5.3 gave a bird's-eye view of the design process, which gives a context for much of the rest of this book.

The process starts with understanding the situation as it is and the requirements for change. Section 5.4 provided some simple techniques for dealing with this: getting to know your users, who they are, remembering that they are different from you, but trying to imagine what it is like for them. You can talk to users, but you should also observe them in other ways, as we are all bad at articulating what we do. One way to help retain a user focus in design is to use personae – detailed word pictures of imaginary but typical users.

Section 5.5 introduced scenarios and rich stories about design, which can help us explore the design space and to discuss potential designs with other designers and potential users. Both scenarios and personae need to be vivid and to include rich contextual details – not just a record of user actions on the system!

The details of potential designs need to be worked out and in Section 5.6 we looked at the overall navigation design of the system. We started by looking at local structure, the way one screen, page or state of an application relates to those it immediately links to. The users need to know where they are, what they can do, what will happen when they do things, and what has happened in the past. This can aid users as they goal seek, or move closer towards their goals without having to necessarily understand completely the whole route there. The global structure of the application is also important. We saw how hierarchy diagrams can give a logical view of an application, which can be used to design menu or site structures. In contrast, the user dialog focusses on the flow of user and system actions. One way to do this is using network diagrams of screens or states of the system and how they link to one another. Any designed system must also relate to its environment: other applications, other websites, other physical devices.

In Section 5.7 we looked at screen design and layout. We saw that there were various visual tools that could help us to ensure that the physical structure of our screen emphasized the logical structure of the user interaction. These tools included physical grouping, ordering of items, decoration such as fonts, lines and color, alignment and the use of white space. These are important both for appropriate display of information and to lay out controls and data entry fields for ease of use. It is also important that controls have appropriate affordances – that is have visual and tactile attributes that suggest their use. Information presented on screen, whether individual items, tabular or graphical, should be appropriate to the user's purpose and this may mean allowing interactions to change the layout, for example re-sort tables by different columns. Aesthetics are also important, but may conflict with utility. Depending on the context you may need to make different trade-offs between these. Good graphical design is an area and a skill all of its own, but some features such as bad use of color and 3D effects are bad for both aesthetics and usability!

Finally, in Section 5.8, we saw that iteration is an essential part of virtually any interaction design process because we cannot get things right first time. However, iterative methods may get trapped in local maxima. To make iterative processes work, we need either extensive personal experience or theoretical understanding to help us get better initial designs.

EXERCISES

5.1 Use a pocket alarm clock or wristwatch to set yourself alarms every 15 minutes one working day. Write down exactly what you are doing. How surprising is it?

Exercises 5.2, 5.3, 5.4 and 5.5 are based around a nuclear reactor scenario on the book website at: /e3/scenario/nuclear/ You will need to read the scenario in order to answer these exercises.

5.2 Comment on the user of color in the Alarm Control, Emergency Shutdown and Emergency Confirm panels (Figure CS.2 – for figures, see the web scenario).

5.3 Comment on the use of layout and other elements in the control panels (Figures CS.1, CS.2 and CS.3), including the way in which various visual elements support or hinder logical grouping and sequence.

5.4 Working through the accident scenario, explain why the various problems arise.

5.5 Suggest potential ways of improving the interface to avoid a similar problem recurring.

RECOMMENDED READING

J. Preece, Y. Rogers and H. Sharp, *Interaction Design: Beyond Human–Computer Interaction*, John Wiley, 2002.
A general textbook on interaction design with especially strong focus on evaluation.

J. Carroll, editor, *Interacting with Computers*, Vol. 13, No. 1, special issue on 'Scenario-based system development', 2000.
Contributions from several authors on using scenarios in design.

J. Carroll, *Making Use: Scenario-Based Design of Human–Computer Interactions*, MIT Press, 2000.
John Carroll's own book dedicated solely to using scenarios in design.

J. McGrenere and W. Ho, Affordances: clarifying and evolving a concept, *Proceedings of Graphics Interface 2000*, pp. 179–86, 2000.
This paper reviews all the major work on affordances from Gibson's original definition and focusses especially on Norman's popularization of the word which has been the way many encounter it. It also reviews the work of Bill Gaver, who is probably the first person to use affordance as a concept within HCI.

E. Tufte, *Envisioning Information*, Graphics Press, Cheshire, USA, 1990, and
E. Tufte, *The Visual Display of Quantitative Information*, Graphics Press, Cheshire, USA, 1997.
Tufte's books are the 'must read' for graphical presentation, packed with examples and pictures – from timetables to Napoleon's disastrous Russian campaign.

HCI IN THE SOFTWARE PROCESS

6

OVERVIEW

- Software engineering provides a means of understanding the structure of the design process, and that process can be assessed for its effectiveness in interactive system design.
- Usability engineering promotes the use of explicit criteria to judge the success of a product in terms of its usability.
- Iterative design practices work to incorporate crucial customer feedback early in the design process to inform critical decisions which affect usability.
- Design involves making many decisions among numerous alternatives. Design rationale provides an explicit means of recording those design decisions and the context in which the decisions were made.

6.1 INTRODUCTION

In Chapter 4 we concentrated on identifying aspects of usable interactive systems by means of concrete examples of successful paradigms. The design goal is to provide reliable techniques for the repeated design of successful and usable interactive systems. It is therefore necessary that we go beyond the exercise of identifying paradigms and examine the process of interactive system design. In the previous chapter we introduced some of the elements of a user-centered design process. Here we expand on that process, placing the design of interactive systems within the established frameworks of software development.

Within computer science there is already a large subdiscipline that addresses the management and technical issues of the development of software systems – called *software engineering*. One of the cornerstones of software engineering is the *software life cycle*, which describes the activities that take place from the initial concept formation for a software system up until its eventual phasing out and replacement. This is not intended to be a software engineering textbook, so it is not our major concern here to discuss in depth all of the issues associated with software engineering and the myriad life-cycle models.

The important point that we would like to draw out is that issues from HCI affecting the usability of interactive systems are relevant within all the activities of the software life cycle. Therefore, software engineering for interactive system design is not simply a matter of adding one more activity that slots in nicely with the existing activities in the life cycle. Rather, it involves techniques that span the entire life cycle.

We will begin this chapter by providing an introduction to some of the important concepts of software engineering, in Section 6.2. Specifically, we will describe the major activities within the traditional software life cycle and discuss the issues raised by the special needs of interactive systems. We will then describe some specific approaches to interactive system design, which are used to promote product usability throughout the life cycle. In Section 6.3, we will discuss a particular methodology called *usability engineering* in which explicit usability requirements are used as goals for the design process. In Section 6.4, we consider iterative design practices that involve prototyping and participative evaluation. We conclude this chapter with a discussion of *design rationale*. Design is a decision-making activity and it is important to keep track of the decisions that have been made and the context in which they were made. Various design rationale techniques, presented in Section 6.5, are used to support this critical activity.

6.2 THE SOFTWARE LIFE CYCLE

One of the claims for software development is that it should be considered as an engineering discipline, in a way similar to how electrical engineering is considered for hardware development. One of the distinguishing characteristics of any engineering discipline is that it entails the structured application of scientific techniques to the development of some product. A fundamental feature of software engineering, therefore, is that it provides the structure for applying techniques to develop software systems. The software life cycle is an attempt to identify the activities that occur in software development. These activities must then be ordered in time in any development project and appropriate techniques must be adopted to carry them through.

In the development of a software product, we consider two main parties: the customer who requires the use of the product and the designer who must provide the product. Typically, the customer and the designer are groups of people and some people can be both customer and designer. It is often important to distinguish between the customer who is the client of the designing company and the customer who is the eventual user of the system. These two roles of customer can be played by different people. The group of people who negotiate the features of the intended system with the designer may never be actual users of the system. This is often particularly true of web applications. In this chapter, we will use the term 'customer' to refer to the group of people who interact with the design team and we will refer to those who will interact with the designed system as the user or end-user.

6.2.1 Activities in the life cycle

A more detailed description of the life cycle activities is depicted in Figure 6.1. The graphical representation is reminiscent of a waterfall, in which each activity naturally leads into the next. The analogy of the waterfall is not completely faithful to the real relationship between these activities, but it provides a good starting point for discussing the logical flow of activity. We describe the activities of this waterfall model of the software life cycle next.[1]

Requirements specification

In requirements specification, the designer and customer try to capture a description of *what* the eventual system will be expected to provide. This is in contrast to determining *how* the system will provide the expected services, which is the concern of later activities. Requirements specification involves eliciting information from the customer about the work environment, or domain, in which the final product will function. Aspects of the work domain include not only the particular functions that the software product must perform but also details about the environment in which it must operate, such as the people whom it will potentially affect and the new product's relationship to any other products which it is updating or replacing.

Requirements specification begins at the start of product development. Though the requirements are from the customer's perspective, if they are to be met by the

1 Some authors distinguish between the software development process and the software life cycle, the waterfall model being used to describe the former and not the latter. The main distinction for our purposes is that operation and maintenance of the product is not part of the development process.

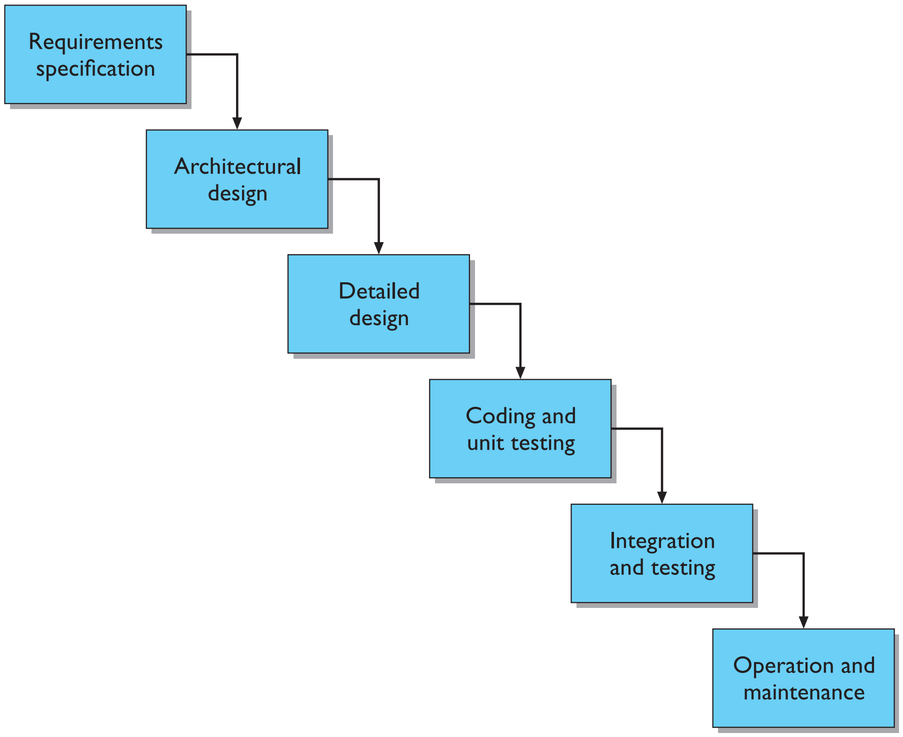


Figure 6.1 The activities in the waterfall model of the software life cycle

software product they must be formulated in a language suitable for implementation. Requirements are usually initially expressed in the native language of the customer. The executable languages for software are less natural and are more closely related to a mathematical language in which each term in the language has a precise interpretation, or semantics. The transformation from the expressive but relatively ambiguous natural language of requirements to the more precise but less expressive executable languages is one key to successful development. In Chapter 15 we discuss task analysis techniques, which are used to express work domain requirements in a form that is both expressive and precise.

Architectural design

As we mentioned, the requirements specification concentrates on what the system is supposed to do. The next activities concentrate on *how* the system provides the services expected from it. The first activity is a high-level decomposition of the system into components that can either be brought in from existing software products or be developed from scratch independently. An architectural design performs this decomposition. It is not only concerned with the functional decomposition of the system, determining which components provide which services. It must also describe the interdependencies between separate components and the sharing of resources that will arise between components.

There are many structured techniques that are used to assist a designer in deriving an architectural description from information in the requirements specification (such as CORE, MASCOT and HOOD). Details of these techniques are outside the scope of this book, but can be found in any good software engineering textbook. What we will mention here is that the majority of these techniques are adequate for capturing the *functional requirements* of the system – the services the system must provide in the work domain – but do not provide an immediate way to capture other *non-functional requirements* – features of the system that are not directly related to the actual services provided but relate to the manner in which those services must be provided. Some classic examples of non-functional requirements are the efficiency, reliability, timing and safety features of the system. Interactive features of the system, such as those that will be described by the principles in Chapter 7, also form a large class of non-functional requirements.

Detailed design

The architectural design provides a decomposition of the system description that allows for isolated development of separate components which will later be integrated. For those components that are not already available for immediate integration, the designer must provide a sufficiently detailed description so that they may be implemented in some programming language. The detailed design is a *refinement* of the component description provided by the architectural design. The behavior implied by the higher-level description must be preserved in the more detailed description.

Typically, there will be more than one possible refinement of the architectural component that will satisfy the behavioral constraints. Choosing the best refinement is often a matter of trying to satisfy as many of the non-functional requirements of the system as possible. Thus the language used for the detailed design must allow some analysis of the design in order to assess its properties. It is also important to keep track of the design options considered, the eventual decisions that were made and the reasons why, as we will discuss in Section 6.5 on design rationale.

Coding and unit testing

The detailed design for a component of the system should be in such a form that it is possible to implement it in some executable programming language. After coding, the component can be tested to verify that it performs correctly, according to some test criteria that were determined in earlier activities. Research on this activity within the life cycle has concentrated on two areas. There is plenty of research that is geared towards the automation of this coding activity directly from a low-level detailed design. Most of the work in *formal methods* operates under the hypothesis that, in theory, the transformation from the detailed design to the implementation is from one mathematical representation to another and so should be able to be entirely automated. Other, more practical work concentrates on the automatic generation of tests from output of earlier activities which can be performed on a piece of code to verify that it behaves correctly.

Integration and testing

Once enough components have been implemented and individually tested, they must be integrated as described in the architectural design. Further testing is done to ensure correct behavior and acceptable use of any shared resources. It is also possible at this time to perform some acceptance testing with the customers to ensure that the system meets their requirements. It is only after acceptance of the integrated system that the product is finally released to the customer.

It may also be necessary to certify the final system according to requirements imposed by some outside authority, such as an aircraft certification board. As of 1993, a European health and safety act requires that all employers provide their staff with usable systems. The international standards authority, ISO, has also produced a standard (ISO 9241) to define the usability of office environment workstations. Coupled together, the health and safety regulations and ISO 9241 provide impetus for designers to take seriously the HCI implications of their design.

Maintenance

After product release, all work on the system is considered under the category of maintenance, until such time as a new version of the product demands a total redesign or the product is phased out entirely. Consequently, the majority of the lifetime of a product is spent in the maintenance activity. Maintenance involves the correction of errors in the system which are discovered after release and the revision of the system services to satisfy requirements that were not realized during previous development. Therefore, maintenance provides feedback to all of the other activities in the life cycle, as shown in Figure 6.2.

6.2.2 Validation and verification

Throughout the life cycle, the design must be checked to ensure that it both satisfies the high-level requirements agreed with the customer and is also complete and internally consistent. These checks are referred to as *validation* and *verification*, respectively. Boehm [36a] provides a useful distinction between the two, characterizing validation as designing 'the right thing' and verification as designing 'the thing right'. Various languages are used throughout design, ranging from informal natural language to very precise and formal mathematical languages. Validation and verification exercises are difficult enough when carried out within one language; they become much more difficult, if not impossible, when attempted between languages.

Verification of a design will most often occur within a single life-cycle activity or between two adjacent activities. For example, in the detailed design of a component

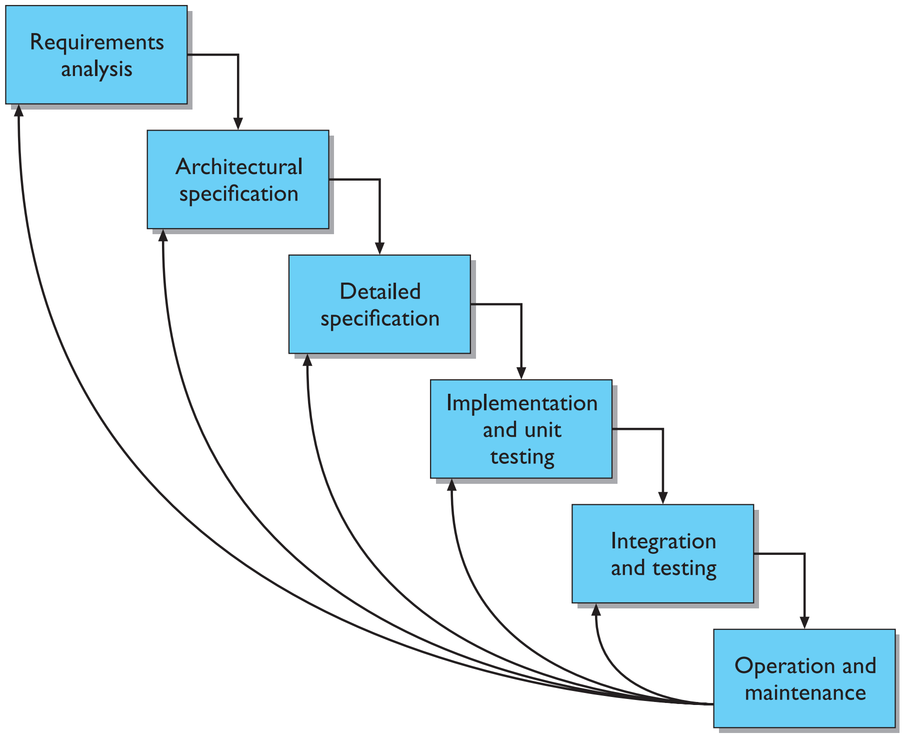

Requirements analysis
Architectural specification
Detailed specification
Implementation and unit testing
Integration and testing
Operation and maintenance

Figure 6.2 Feedback from maintenance activity to other design activities

of a payroll accounting system, the designer will be concerned with the correctness of the algorithm to compute taxes deducted from an employee's gross income. The architectural design will have provided a general specification of the information input to this component and the information it should output. The detailed description will introduce more information in refining the general specification. The detailed design may also have to change the representations for the information and will almost certainly break up a single high-level operation into several low-level operations that can eventually be implemented. In introducing these changes to information and operations, the designer must show that the refined description is a legal one within its language (internal consistency) and that it describes all of the *specified* behavior of the high-level description (completeness) in a provably correct way (relative consistency).

Validation of a design demonstrates that within the various activities the customer's requirements are satisfied. Validation is a much more subjective exercise than verification, mainly because the disparity between the language of the requirements and the language of the design forbids any objective form of proof. In interactive system design, the validation against HCI requirements is often referred to as evaluation and can be performed by the designer in isolation or in cooperation with the customer. We discuss evaluation in depth in Chapter 9.

An important question, which applies to both verification and validation, asks exactly what constitutes a proof. We have repeatedly mentioned the language used in any design activity and the basis for the semantics of that language. Languages with a mathematical foundation allow reasoning and proof in the objective sense. An argument based entirely within some mathematical language can be accepted or refuted based upon universally accepted measures. A proof can be entirely justified by the rules of the mathematical language, in which case it is considered a formal proof. More common is a rigorous proof, which is represented within some mathematical language but which relies on the understanding of the reader to accept its correctness without appeal to the full details of the argument, which could be provided but usually are not. The difference between formality and rigour is in the amount of detail the prover leaves out while still maintaining acceptance of the proof.

Proofs that are for verification of a design can frequently occur within one language or between two languages which both have a precise mathematical semantics. Time constraints for a design project and the perceived economic implications of the separate components usually dictate which proofs are carried out in full formality and which are done only rigorously (if at all). As research in this area matures and automated tools provide assistance for the mechanical aspects of proof, the cost of proof should decrease.

Validation proofs are much trickier, as they almost always involve a transformation between languages. Furthermore, the origin of customer requirements arises in the inherent ambiguity of the real world and not the mathematical world. This precludes the possibility of objective proof, rigorous or formal. Instead, there will always be a leap from the informal situations of the real world to any formal and structured development process. We refer to this inevitable disparity as the *formality gap*, depicted in Figure 6.3.

The formality gap means that validation will always rely to some extent on subjective means of proof. We can increase our confidence in the subjective proof by effective use of real-world experts in performing certain validation chores. These experts will not necessarily have design expertise, so they may not understand the

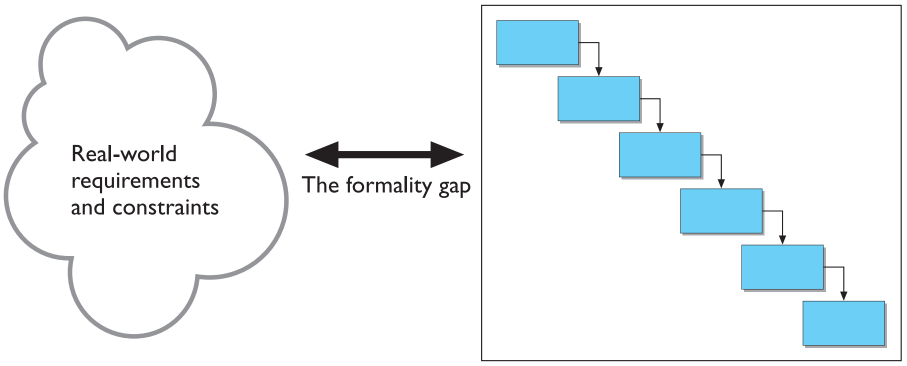


Figure 6.3 The formality gap between the real world and structured design

design notations used. Therefore, it is important that the design notations narrow the formality gap, making clear the claims that the expert can then validate. For interactive systems, the expert will have knowledge from a cognitive or psychological domain, so the design specification must be readily interpretable from a psychological perspective in order to validate it against interactive requirements of the system. We will discuss design techniques and notations that narrow the formality gap for validation of interactive properties of systems in Part 3.

6.2.3 Management and contractual issues

The life cycle described above concentrated on the more technical features of software development. In a technical discussion, managerial issues of design, such as time constraints and economic forces, are not as important. The different activities of the life cycle are logically related to each other. We can see that requirements for a system precede the high-level architectural design which precedes the detailed design, and so on. In reality, it is quite possible that some detailed design is attempted before all of the architectural design. In management, a much wider perspective must be adopted which takes into account the marketability of a system, its training needs, the availability of skilled personnel or possible subcontractors, and other topics outside the activities for the development of the isolated system.

As an example, we will take the development of a new aircraft on which there will be many software subsystems. The aircraft company will usually go through a concept evaluation period of up to 10 years before making any decision about actual product development. Once it has been decided to build a certain type of aircraft, loosely specified in the case of commercial aircraft in terms of passenger capacity and flight range, more explicit design activity follows. This includes joint analysis for both the specification of the aircraft and determination of training needs. It is only after the architectural specification of the aircraft is complete that the separate systems to be developed are identified. Some of these systems will be software systems, such as the flight management system or the training simulator, and these will be designed according to the life cycle described earlier. Typically, this will take four to five years. The separate aircraft systems are then integrated for ground and flight testing and certification before the aircraft is delivered to any customer airlines. The operating lifetime of an aircraft model is expected to be in the range of 20–40 years, during which time maintenance must be provided. The total lifetime of an aircraft from conception to phasing out is up to 55 years, only 4–5 years (excluding maintenance) of which contain the software life cycle which we are discussing in this chapter.

In managing the development process, the temporal relationship between the various activities is more important, as are the intermediate deliverables which represent the technical content, as the designer must demonstrate to the customer that progress is being made. A useful distinction, taken from McDermid [232], is that the technical perspective of the life cycle is described in *stages* of activity, whereas the managerial perspective is described in temporally bound *phases*. A phase is usually defined in terms of the documentation taken as input to the phase and the documentation delivered as output from the phase. So the requirements phase will take any marketing or conceptual development information, identifying potential customers, as input and produce a requirements specification that must be agreed upon between customer and designer.

This brings up another important issue from the management perspective. As the design activity proceeds, the customer and the designer must sign off on various documents, indicating their satisfaction with progress to date. These signed documents can carry a varying degree of contractual obligation between customer and designer. A signed requirements specification indicates both that the customer agrees to limit demands of the eventual product to those listed in the specification and also that the designer agrees to meet all of the requirements listed. From a technical perspective, it is easy to acknowledge that it is difficult, if not impossible, to determine all of the requirements before embarking on any other design activity. A satisfactory requirements specification may not be known until after the product has been in operation! From a management perspective, it is unacceptable to both designer and customer to delay the requirements specification that long.

So contractual obligation is a necessary consequence of managing software development, but it has negative implications on the design process as well. It is very difficult in the design of an interactive system to determine a priori what requirements to impose on the system to maximize its usability. Having to fix on some requirements too early will result either in general requirements that are very little guide for the designer or in specific requirements that compromise the flexibility of design without guaranteeing any benefits.

6.2.4 Interactive systems and the software life cycle

The traditional software engineering life cycles arose out of a need in the 1960s and 1970s to provide structure to the development of large software systems. In those days, the majority of large systems produced were concerned with data-processing applications in business. These systems were not highly interactive; rather, they were batch-processing systems. Consequently, issues concerning usability from an end-user's perspective were not all that important. With the advent of personal computing in the late 1970s and its huge commercial success and acceptance, most modern systems developed today are much more interactive, and it is vital to the success of any product that it be easy to operate for someone who is not expected to know much about how the system was designed. The modern user has a great amount of skill in the work that he performs without necessarily having that much skill in software development.

The life cycle for development we described above presents the process of design in a somewhat pipeline order. In reality, even for batch-processing systems, the actual design process is *iterative*, work in one design activity affecting work in any other activity both before or after it in the life cycle. We can represent this iterative relationship as in Figure 6.4, but that does not greatly enhance any understanding of the design process for interactive systems. You may ask whether it is worth the

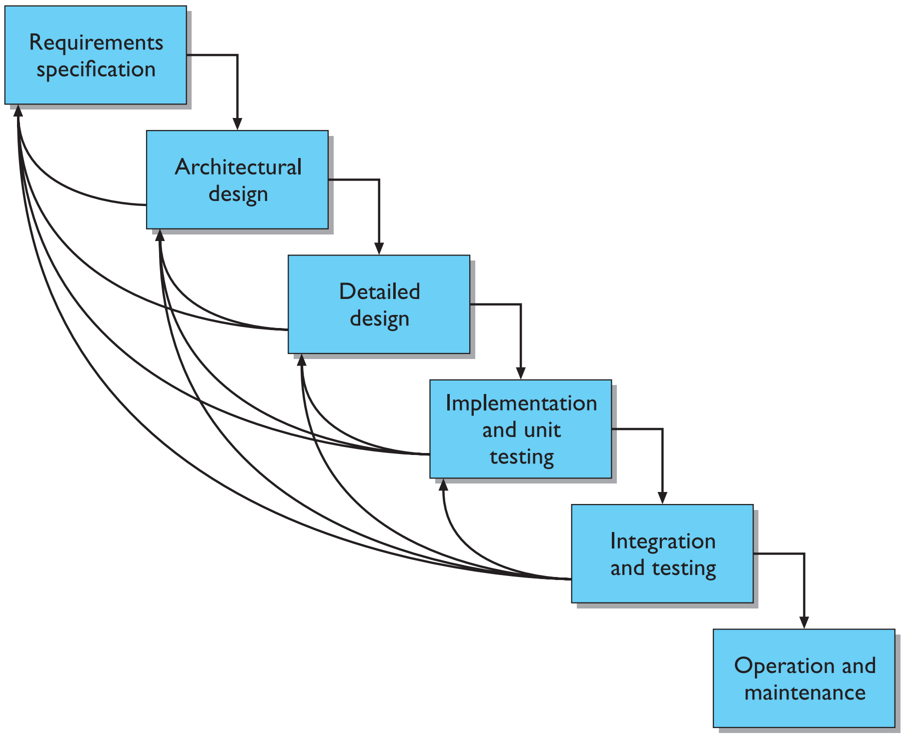

Requirements specification
Architectural design
Detailed design
Implementation and unit testing
Integration and testing
Operation and maintenance

Figure 6.4 Representing iteration in the waterfall model

intellectual effort to understand the interactive system design process. Is there really much design effort spent on the interactive aspects of a system to warrant our attention? A classic survey in 1978 by Sutton and Sprague at IBM resulted in an estimate that 50% of the designer's time was spent on designing code for the user interface [338]. A more recent and convincing survey by Myers and Rosson has confirmed that that finding holds true for the 1990s [247]. So it is definitely worth the effort to provide structure and techniques to understand, structure and improve the interactive design process! In this section, we will address features of interactive system design which are not treated properly by the traditional software life cycle.

The traditional software life cycle suits a principled approach to design; that is, if we know what it is we want to produce from the beginning, then we can structure our approach to design in order to attain the goal. We have already mentioned how, in practice, designers do not find out all of the requirements for a system before they begin. Figure 6.4 depicts how discovery in later activities can be reflected in iterations back to earlier stages. This is an admission that the requirements capture activity is not executed properly. The more serious claim we are making here is that all of the requirements for an interactive system *cannot* be determined from the start, and there are many convincing arguments to support this position. The result is that systems must be built and the interaction with users observed and evaluated in order to determine how to make them more usable.

Our models of the psychology and sociology of the human and human cognition, whether in isolation or in a group, are incomplete and do not allow us to predict how to design for maximum usability. There is much research on models of human users that allow prediction of their performance with interactive systems, which we will discuss in Chapter 12. These models, however, either rely on too much detail of the system to be useful at very early and abstract stages of design (see the section in Chapter 12 on the keystroke-level model) or they only apply to goal-oriented planned activity and not highly interactive WIMP systems (refer to the discussion at the end of Chapter 12).

This dearth of predictive psychological theory means that in order to test certain usability properties of their designs, designers must observe how actual users interact with the developed product and measure their performance. In order for the results of those observations to be worthwhile, the experiments must be as close to a real interaction situation as possible. That means the experimental system must be very much like it would be in the final product whose requirements the designer is trying to establish! As John Carroll has pointed out, the very detail of the actual system can crucially affect its usability, so it is not worthwhile to experiment on crude estimates of it, as that will provide observations whose conclusions will not necessarily apply to the real system [59].

One principled approach to interactive system design, which will be important in later chapters, relies on a clear understanding early on in the design of the tasks that the user wishes to perform. One problem with this assumption is that the tasks a user will perform are often only known by the user after he is familiar with the system on which he performs them. The chicken-and-egg puzzle applies to tasks and the artifacts on which he performs those tasks. For example, before the advent of word processors, an author would not have considered the use of a contracting and expanding outlining facility to experiment easily and quickly with the structure of a paper while it was being typed. A typewriter simply did not provide the ability to perform such a task, so how would a designer know to support such a task in designing the first word processor?

Also, some of the tasks a user performs with a system were never explicitly intended as tasks by its designer. Take the example of a graphics drawing package that separates the constructed picture into separate layers. One layer is used to build graphical pictures which are entire objects – a circle or a square, for instance – and can be manipulated as those objects and retain their object identity. The other layer is used to paint pictures which are just a collection of pixels. The user can switch between the layers in order to create very complex pictures which are part object, part painted scene. But because of the complex interplay between overlapping images between the two layers, it is also possible to hide certain parts of the picture when in one layer and reveal them in the other layer. Such a facility will allow the user to do simple simulations, such as showing the effect of shadowing when switching a light on and off. It is very doubtful that the designers were thinking explicitly of supporting such simulation or animation tasks when they were designing these graphics systems, which were meant to build complex, but static, pictures.

A final point about the traditional software life cycle is that it does not promote the use of notations and techniques that support the user's perspective of the interactive system. We discussed earlier the purpose of validation and the formality gap. It is very difficult for an expert on human cognition to predict the cognitive demands that an abstract design would require of the intended user if the notation for the design does not reflect the kind of information the user must recall in order to interact. The same holds for assessing the timing behavior of an abstract design that does not explicitly mention the timing characteristics of the operations to be invoked or their relative ordering. Though no structured development process will entirely eliminate the formality gap, the particular notations used can go a long way towards making validation of non-functional requirements feasible with expert assistance.

In the remaining sections of this chapter, we will describe various approaches to augment the design process to suit better the design of interactive systems. These approaches are categorized under the banner of *user-centered design*.

6.3 USABILITY ENGINEERING

One approach to user-centered design has been the introduction of explicit *usability engineering* goals into the design process, as suggested by Whiteside and colleagues at IBM and Digital Equipment Corporation [377] and by Nielsen at Bellcore [260, 261]. Engineering depends on interpretation against a shared background of meaning, agreed goals and an understanding of how satisfactory completion will be judged. The emphasis for usability engineering is in knowing exactly what criteria will be used to judge a product for its usability.

The ultimate test of a product's usability is based on measurements of users' experience with it. Therefore, since a user's direct experience with an interactive system is at the physical interface, focus on the actual user interface is understandable. The danger with this limited focus is that much of the work that is accomplished in interaction involves more than just the surface features of the systems used to perform that work. In reality, the whole functional architecture of the system and the cognitive capacity of the users should be observed in order to arrive at meaningful measures. But it is not at all simple to derive measurements of activity beyond the physical actions in the world, and so usability engineering is limited in its application.

In relation to the software life cycle, one of the important features of usability engineering is the inclusion of a usability specification, forming part of the requirements specification, that concentrates on features of the user–system interaction which contribute to the usability of the product. Various attributes of the system are suggested as gauges for testing the usability. For each attribute, six items are defined to form the usability specification of that attribute. Table 6.1 provides an example of a usability specification for the design of a control panel for a video cassette recorder (VCR), based on the technique presented by Whiteside, Bennett and Holtzblatt [377].

Table 6.1 Sample usability specification for undo with a VCR

Attribute:	Backward recoverability
Measuring concept:	Undo an erroneous programming sequence
Measuring method:	Number of explicit user actions to undo current program
Now level:	No current product allows such an undo
Worst case:	As many actions as it takes to program in mistake
Planned level:	A maximum of two explicit user actions
Best case:	One explicit cancel action

In this example, we choose the principle of recoverability, described fully in Chapter 7, as the particular usability attribute of interest. Recoverability refers to the ability to reach a desired goal after recognition of some error in previous interaction. The recovery procedure can be in either a backward or forward sense. Current VCR design has resulted in interactive systems that are notoriously difficult to use; the redesign of a VCR provides a good case study for usability engineering. In designing a new VCR control panel, the designer wants to take into account how a user might recover from a mistake he discovers while trying to program the VCR to record some television program in his absence. One approach that the designer decides to follow is to allow the user the ability to undo the programming sequence, reverting the state of the VCR to what it was before the programming task began.

The backward recoverability attribute is defined in terms of a *measuring concept*, which makes the abstract attribute more concrete by describing it in terms of the actual product. So in this case, we realize backward recoverability as the ability to undo an erroneous programming sequence. The *measuring method* states how the attribute will be measured, in this case by the number of explicit user actions required to perform the undo, regardless of where the user is in the programming sequence.

The remaining four entries in the usability specification then provide the agreed criteria for judging the success of the product based on the measuring method. The *now level* indicates the value for the measurement with the existing system, whether it is computer based or not. The *worst case* value is the lowest acceptable measurement for the task, providing a clear distinction between what will be acceptable and what will be unacceptable in the final product. The *planned level* is the target for the design and the *best case* is the level which is agreed to be the best possible measurement given the current state of development tools and technology.

In the example, the designers can look at their previous VCR products and those of their competitors to determine a suitable now level. In this case, it is determined that no current model allows an undo which returns the state of the VCR to what it was before the programming task. For example, if a VCR allows you three separate recording programs, once you begin entering a new program in the number 1 program slot, the VCR forgets the previous contents of that slot and so you cannot recover it unless you remember what it was and then reprogram it.

Table 6.2 Criteria by which measuring method can be determined (adapted from Whiteside, Bennett and Holtzblatt [377], Copyright 1988, reprinted with permission from Elsevier)

1.	Time to complete a task
2.	Per cent of task completed
3.	Per cent of task completed per unit time
4.	Ratio of successes to failures
5.	Time spent in errors
6.	Per cent or number of errors
7.	Per cent or number of competitors better than it
8.	Number of commands used
9.	Frequency of help and documentation use
10.	Per cent of favorable/unfavorable user comments
11.	Number of repetitions of failed commands
12.	Number of runs of successes and of failures
13.	Number of times interface misleads the user
14.	Number of good and bad features recalled by users
15.	Number of available commands not invoked
16.	Number of regressive behaviors
17.	Number of users preferring your system
18.	Number of times users need to work around a problem
19.	Number of times the user is disrupted from a work task
20.	Number of times user loses control of the system
21.	Number of times user expresses frustration or satisfaction

Determining the worst case value depends on a number of things. Usually, it should be no lower than the now level. The new product should provide some improvement on the current state of affairs, and so it seems that at least some of the usability attributes should provide worst case values that are better than the now level. Otherwise, why would the customer bother with the new system (unless it can be shown to provide the same usability at a fraction of the cost)? The designers in the example have determined that the minimal acceptable undo facility would require the user to perform as many actions as he had done to program in the mistake. This is a clear improvement over the now level, since it at least provides for the possibility of undo. One way to provide such a capability would be by including an undo button on the control panel, which would effectively reverse the previous non-undo action. The designers figure that they should allow for the user to do a complete restoration of the VCR state in a maximum of two explicit user actions, though they recognize that the best case, at least in terms of the number of explicit actions, would require only one.

Tables 6.2 and 6.3, adapted from Whiteside, Bennett and Holtzblatt [377], provide a list of measurement criteria which can be used to determine the measuring method for a usability attribute and the possible ways to set the worst/best case and planned/now level targets. Measurements such as those promoted by usability engineering are also called *usability metrics*.

Table 6.3 Possible ways to set measurement levels in a usability specification (adapted from Whiteside, Bennett and Holtzblatt [377], Copyright 1988, reprinted with permission from Elsevier)

Set levels with respect to information on:
1. an existing system or previous version
2. competitive systems
3. carrying out the task without use of a computer system
4. an absolute scale
5. your own prototype
6. user's own earlier performance
7. each component of a system separately
8. a successive split of the difference between best and worst values observed in user tests

Table 6.4 Examples of usability metrics from ISO 9241

Usability objective	Effectiveness measures	Efficiency measures	Satisfaction measures
Suitability for the task	Percentage of goals achieved	Time to complete a task	Rating scale for satisfaction
Appropriate for trained users	Number of power features used	Relative efficiency compared with an expert user	Rating scale for satisfaction with power features
Learnability	Percentage of functions learned	Time to learn criterion	Rating scale for ease of learning
Error tolerance	Percentage of errors corrected successfully	Time spent on correcting errors	Rating scale for error handling

The ISO standard 9241, described earlier, also recommends the use of usability specifications as a means of requirements specification. Table 6.4 gives examples of usability metrics categorized by their contribution towards the three categories of usability: effectiveness, efficiency and satisfaction.

6.3.1 Problems with usability engineering

The major feature of usability engineering is the assertion of explicit usability metrics early on in the design process which can be used to judge a system once it is delivered. There is a very solid argument which points out that it is only through empirical approaches such as the use of usability metrics that we can reliably build more usable systems. Although the ultimate yardstick for determining usability may be by observing and measuring user performance, that does not mean that these measurements are the best way to produce a predictive design process for usability.

The problem with usability metrics is that they rely on measurements of very specific user actions in very specific situations. When the designer knows what the actions and situation will be, then she can set goals for measured observations. However, at early stages of design, designers do not have this information. Take our example usability specification for the VCR. In setting the acceptable and unacceptable levels for backward recovery, there is an assumption that a button will be available to invoke the undo. In fact, the designer was already making an implicit assumption that the user would be making errors in the programming of the VCR. Why not address the origin of the programming errors, then maybe undo would not be necessary?

We should recognize another inherent limitation for usability engineering, that is it provides a means of satisfying usability specifications and not necessarily usability. The designer is still forced to understand why a particular usability metric enhances usability for real people. Again, in the VCR example, the designer assumed that fewer explicit actions make the undo operation easier. Is that kind of assumption warranted?

6.4 ITERATIVE DESIGN AND PROTOTYPING

A point we raised earlier is that requirements for an interactive system cannot be completely specified from the beginning of the life cycle. The only way to be sure about some features of the potential design is to build them and test them out on real users. The design can then be modified to correct any false assumptions that were revealed in the testing. This is the essence of *iterative design*, a purposeful design process which tries to overcome the inherent problems of incomplete requirements specification by cycling through several designs, incrementally improving upon the final product with each pass.

The problems with the design process, which lead to an iterative design philosophy, are not unique to the usability features of the intended system. The problem holds for requirements specification in general, and so it is a general software engineering problem, together with technical and managerial issues.

On the technical side, iterative design is described by the use of *prototypes*, artifacts that simulate or animate some but not all features of the intended system. There are three main approaches to prototyping:

Throw-away The prototype is built and tested. The design knowledge gained from this exercise is used to build the final product, but the actual prototype is discarded. Figure 6.5 depicts the procedure in using throw-away prototypes to arrive at a final requirements specification in order for the rest of the design process to proceed.

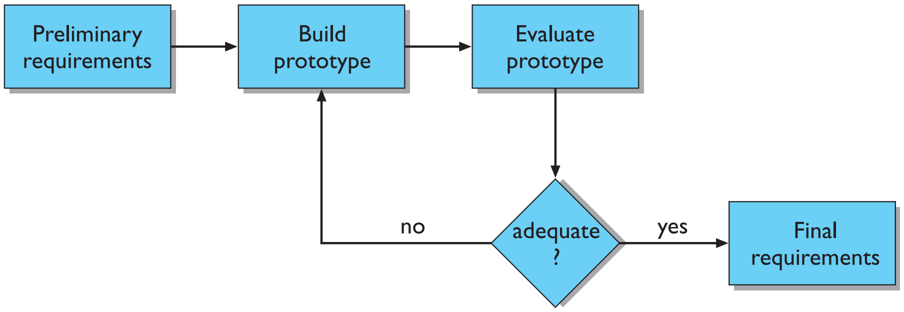


Figure 6.5 Throw-away prototyping within requirements specification

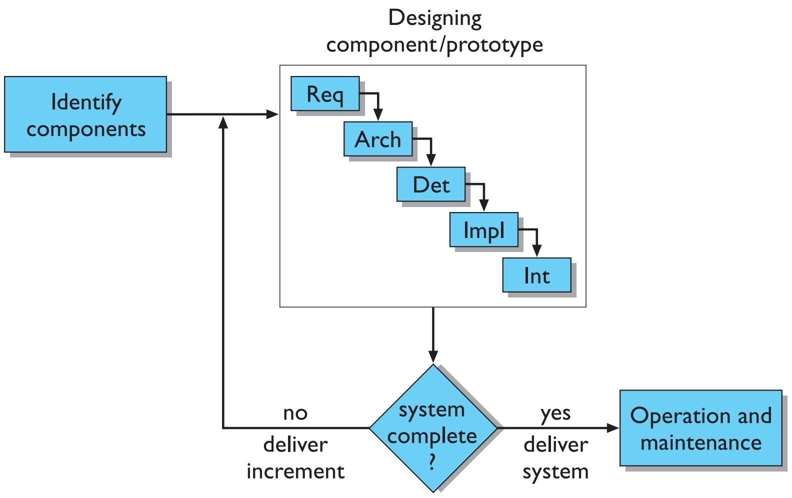


Figure 6.6 Incremental prototyping within the life cycle

Incremental The final product is built as separate components, one at a time. There is one overall design for the final system, but it is partitioned into independent and smaller components. The final product is then released as a series of products, each subsequent release including one more component. This is depicted in Figure 6.6.

Evolutionary Here the prototype is not discarded and serves as the basis for the next iteration of design. In this case, the actual system is seen as evolving from a very limited initial version to its final release, as depicted in Figure 6.7. Evolutionary prototyping also fits in well with the modifications which must be made to the system that arise during the operation and maintenance activity in the life cycle.

Prototypes differ according to the amount of functionality and performance they provide relative to the final product. An *animation* of requirements can involve no

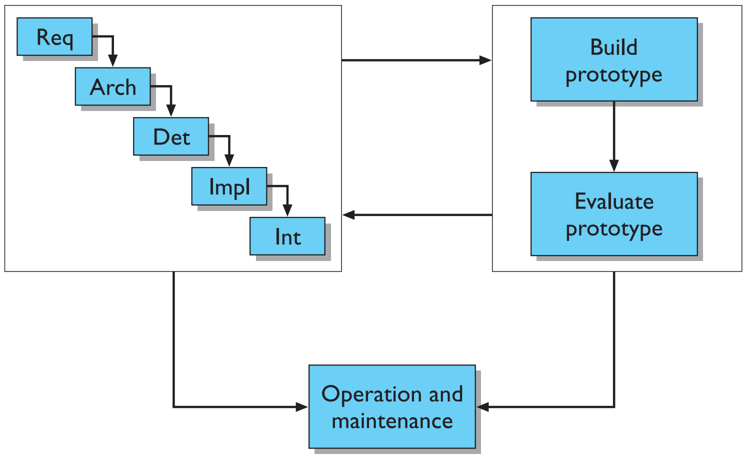


Figure 6.7 Evolutionary prototyping throughout the life cycle

real functionality, or limited functionality to simulate only a small aspect of the interactive behavior for evaluative purposes. At the other extreme, full functionality can be provided at the expense of other performance characteristics, such as speed or error tolerance. Regardless of the level of functionality, the importance of a prototype lies in its projected realism. The prototype of an interactive system is used to test requirements by evaluating their impact with real users. An honest appraisal of the requirements of the final system can only be trusted if the evaluation conditions are similar to those anticipated for the actual operation. But providing realism is costly, so there must be support for a designer/programmer to create a realistic prototype quickly and efficiently.

On the management side, there are several potential problems, as pointed out by Sommerville [327]:

Time Building prototypes takes time and, if it is a throw-away prototype, it can be seen as precious time taken away from the real design task. So the value of prototyping is only appreciated if it is fast, hence the use of the term *rapid prototyping*. However, rapid development and manipulation of a prototype should not be mistaken for rushed evaluation which might lead to erroneous results and invalidate the only advantage of using a prototype in the first place.

Planning Most project managers do not have the experience necessary for adequately planning and costing a design process which involves prototyping.

Non-functional features Often the most important features of a system will be non-functional ones, such as safety and reliability, and these are precisely the kinds of features which are sacrificed in developing a prototype. For evaluating usability features of a prototype, response time – yet another feature often compromised in a prototype – could be critical to product acceptance. This problem is similar to the technical issue of prototype realism.

Contracts The design process is often governed by contractual agreements between customer and designer which are affected by many of these managerial and technical issues. Prototypes and other implementations cannot form the basis for a legal contract, and so an iterative design process will still require documentation which serves as the binding agreement. There must be an effective way of translating the results derived from prototyping into adequate documentation. A rapid prototyping process might be amenable to quick changes, but that does not also apply to the design process.

6.4.1 Techniques for prototyping

Here we will describe some of the techniques that are available for producing rapid prototypes.

Storyboards

Probably the simplest notion of a prototype is the *storyboard*, which is a graphical depiction of the outward appearance of the intended system, without any accompanying system functionality. Storyboards do not require much in terms of computing power to construct; in fact, they can be mocked up without the aid of any computing resource. The origins of storyboards are in the film industry, where a series of panels roughly depicts snapshots from an intended film sequence in order to get the idea across about the eventual scene. Similarly, for interactive system design, the storyboards provide snapshots of the interface at particular points in the interaction. Evaluating customer or user impressions of the storyboards can determine relatively quickly if the design is heading in the right direction.

Modern graphical drawing packages now make it possible to create storyboards with the aid of a computer instead of by hand. Though the graphic design achievable on screen may not be as sophisticated as that possible by a professional graphic designer, it is more realistic because the final system will have to be displayed on a screen. Also, it is possible to provide crude but effective *animation* by automated sequencing through a series of snapshots. Animation illustrates the dynamic aspects of the intended user–system interaction, which may not be possible with traditional paper-based storyboards. If not animated, storyboards usually include annotations and scripts indicating how the interaction will occur.

Limited functionality simulations

More functionality must be built into the prototype to demonstrate the work that the application will accomplish. Storyboards and animation techniques are not sufficient for this purpose, as they cannot portray adequately the interactive aspects of the system. To do this, some portion of the functionality must be *simulated* by the design team.

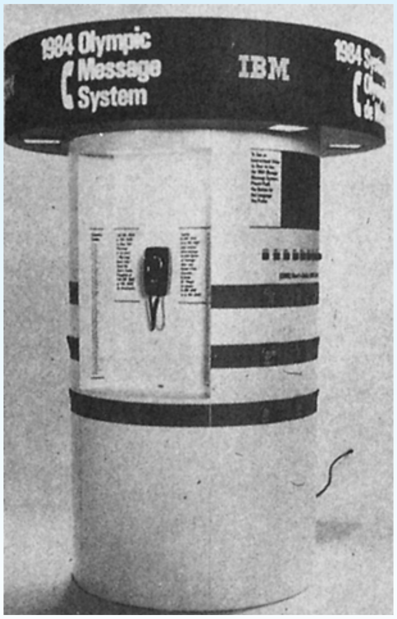
Programming support for simulations means a designer can rapidly build graphical and textual interaction objects and attach some behavior to those objects, which mimics the system's functionality. Once this simulation is built, it can be evaluated and changed rapidly to reflect the results of the evaluation study with various users.

For example, we might want to build a prototype for the VCR with undo described earlier using only a workstation display, keyboard and mouse. We could draw a picture of the VCR with its control panel using a graphics drawing package, but then we would want to allow a subject to use the mouse to position a finger cursor over one of the buttons to 'press' it and actuate some behavior of the VCR. In this way, we could simulate the programming task and experiment with different options for undoing.

DESIGN FOCUS

Prototyping in practice

IBM supplied the computerized information and messaging booths for the 1984 Olympics in Los Angeles. These booths were to be used by the many thousands of residents in the Olympic village who would have to use them with no prior training (extensive instructions in several hundred languages being impractical). IBM sampled several variants on the kiosk design of the telephone-based system, using what they called the hallway and storefront methodology [152]. The final system was intended to be a walk-up-and-use system, so it was important to get comments from people with no knowledge of the process. Early versions of the kiosk were displayed as storyboards on a mock kiosk design in the front hallway of the Yorktown Research Lab. Passers-by were encouraged to browse at the display much as they would a storefront in the window. As casual comments were made and the kiosk was modified according to those comments, more and more active evaluation was elicited. This procedure helped to determine the ultimate positioning of display screens and telephones for the final design.



An Olympic Message System Kiosk (Gould J. D., Boies S. J., Levy S., Richards J. T. and Schoonard J. (1987). The 1984 Olympic Message System: a test of behavioral principles of system design. *Communications of the ACM*, **30**(9), 758–69. Copyright © 1987 ACM, Inc. Reprinted by permission)

There are now plenty of prototyping tools available which allow the rapid development of such simulation prototypes. These simulation tools are meant to provide a quick development process for a very wide range of small but highly interactive applications. A well-known and successful prototyping tool is *HyperCard*, a simulation environment for the Macintosh line of Apple computers. HyperCard is similar to the animation tools described above in that the user can create a graphical depiction of some system, say the VCR, with common graphical tools. The graphical images are placed on cards, and links between cards can be created which control the sequencing from one card to the next for animation effects. What HyperCard provides beyond this type of animation is the ability to describe more sophisticated interactive behavior by attaching a *script*, written in the HyperTalk programming language, to any object. So for the VCR, we could attach a script to any control panel button to highlight it or make an audible noise when the user clicks the mouse cursor over it. Then some functionality could be associated to that button by reflecting some change in the VCR display window. Similar functionality is provided through tools such as Macromedia Flash and Director.

Most of the simulations produced are intended to be throw-away prototypes because of their relatively inefficient implementation. They are not intended to support full-blown systems development and they are unsatisfactory in that role. However, as more designers recognize the utility of prototyping and iterative design, they are beginning to demand ways of incorporating the prototypes into the final delivered systems – more along the lines of evolutionary prototyping. A good example of this is in the avionics industry, where it has long been recognized that iterative development via rapid prototyping and evaluation is essential for the design of flight deck instrumentation and controls. Workstation technology provides sufficient graphics capabilities to enable a designer to produce very realistic gauges, which can be assessed and critiqued by actual pilots. With the advent of the glass cockpit – in which traditional mechanical gauges are replaced by gauges represented on video displays – there is no longer a technology gap between the prototype designs of flight deck instruments and the actual instruments in flight. Therefore, it is a reasonable request by these designers that they be able to reuse the functionality of the prototypes in the actual flight simulators and cockpits, and this demand is starting to be met by commercial prototyping systems which produce efficient code for use in such safety-critical applications.

One technique for simulation, which does not require very much computer-supported functionality, is the *Wizard of Oz* technique. With this technique, the designers can develop a limited functionality prototype and enhance its functionality in evaluation by providing the missing functionality through human intervention. A participant in the evaluation of a new accounting system may not have any computer training but is familiar with accounting procedures. He is asked to sit down in front of the prototype accounting system and to perform some task, say to check the accounts receivable against some newly arrived payments. The naïve computer user will not know the specific language of the system, but you do not want him to worry about that. Instead, he is given instructions to type whatever seems the most natural commands to the system. One of the designers – the wizard in this scenario – is situated in another room, out of sight of the subject, but she is able to receive the subject's input commands and translate them into commands that will work on the prototype. By intervening between the user and system, the wizard is able to increase the perceived functionality of the system so that evaluation can concentrate on how the subject would react to the complete system. Examination of how the wizard had to interpret the subject's input can provide advice as to how the prototype must be enhanced in its later versions.

High-level programming support

HyperTalk was an example of a special-purpose high-level programming language which makes it easy for the designer to program certain features of an interactive system at the expense of other system features like speed of response or space efficiency. HyperTalk and many similar languages allow the programmer to attach functional behavior to the specific interactions that the user will be able to do, such as position and click on the mouse over a button on the screen. Previously, the difficulty of interactive programming was that it was so implementation dependent that the programmer would have to know quite a bit of intimate detail of the hardware system in order to control even the simplest of interactive behavior. These high-level programming languages allow the programmer to abstract away from the hardware specifics and think in terms that are closer to the way the input and output devices are perceived as interaction devices.

Though not usually considered together with such simulation environments, a *user interface management system* – or UIMS (pronounced 'you-imz') – can be considered to provide such high-level programming support. The frequent conceptual model put forth for interactive system design is to separate the application functionality from its presentation. It is then possible to program the underlying functionality of the system and to program the behavior of the user interface separately. The job of a UIMS, then, is to allow the programmer to connect the behavior at the interface with the underlying functionality. In Chapter 8 we will discuss in more detail the advantages and disadvantages of such a conceptual model and concentrate on the programming implementation support provided by a UIMS. What is of interest here is that the separation implied by a UIMS allows the independent development of the features of the interface apart from the underlying functionality. If the underlying system is already developed, then various prototypes of its interface can be quickly constructed and evaluated to determine the optimal one.

6.4.2 Warning about iterative design

Though we have presented the process of iterative design as not only beneficial but also necessary for good interactive system design, it is important to recognize some of its drawbacks, in addition to the very real management issues we have already raised. The ideal model of iterative design, in which a rapid prototype is designed, evaluated and modified until the best possible design is achieved in the given project time, is appealing. But there are two problems.

First, it is often the case that design decisions made at the very beginning of the prototyping process are wrong and, in practice, design inertia can be so great as never to overcome an initial bad decision. So, whereas iterative design is, in theory, amenable to great changes through iterations, it can be the case that the initial prototype has bad features that will not be amended. We will examine this problem through a real example of a clock on a microwave oven.[2] The clock has a numeric display of four digits. Thus the display is capable of showing values in the range from `00:00` to `99:99`. The functional model of time for the actual clock is only 12 hours, so quite a few of the possible clock displays do not correspond to possible times (for example, `63:00`, `85:49`), even though some of them are legal four-digit time designations. That poses no problem, as long as both the designer and the ultimate users of the clock both share the knowledge of the discrepancy between possible clock displays and legal times. Such would not be the case for someone assuming a 24-hour time format, in which case the displays `00:30` and `13:45` would represent valid times in their model but not in the microwave's model. In this particular example, the subjects tested during the evaluation must have all shared the 12-hour time model, and the mismatch with the other users (with a 24-hour model) was only discovered after the product was being shipped. At this point, the only impact of iterative design was a change to the documentation alerting the reader to the 12-hour format, as it was too late to perform any hardware change.

The second problem is slightly more subtle, and serious. If, in the process of evaluation, a potential usability problem is diagnosed, it is important to understand the reason for the problem and not just detect the symptom. In the clock example, the designers could have noticed that some subjects with a 24-hour time model were having difficulty setting the time. Say they were trying to set the time for `14:45`, but they were not being allowed to do that. If the designers did not know the subject's goals, they might not detect the 24/12 hour discrepancy. They would instead notice that the users were having trouble setting the time and so they might change the buttons used to set the time instead of other possible changes, such as an analog time dial, or displaying AM or PM on the clock dial to make the 12-hour model more obvious, or to change to a 24-hour clock.

The moral for iterative design is that it should be used in conjunction with other, more principled approaches to interactive system design. These principled approaches are the subject of Part 3 of this book.

6.5 DESIGN RATIONALE

In designing any computer system, many decisions are made as the product goes from a set of vague customer requirements to a deliverable entity. Often it is difficult to recreate the reasons, or rationale, behind various design decisions. *Design*

2 This example has been provided by Harold Thimbleby.

rationale is the information that explains why a computer system is the way it is, including its structural or architectural description and its functional or behavioral description. In this sense, design rationale does not fit squarely into the software life cycle described in this chapter as just another phase or box. Rather, design rationale relates to an activity of both reflection (doing design rationale) and documentation (creating a design rationale) that occurs throughout the entire life cycle.

It is beneficial to have access to the design rationale for several reasons:

- In an explicit form, a design rationale provides a communication mechanism among the members of a design team so that during later stages of design and/or maintenance it is possible to understand what critical decisions were made, what alternatives were investigated (and, possibly, in what order) and the reason why one alternative was chosen over the others. This can help avoid incorrect assumptions later.
- Accumulated knowledge in the form of design rationales for a set of products can be reused to transfer what has worked in one situation to another situation which has similar needs. The design rationale can capture the context of a design decision in order that a different design team can determine if a similar rationale is appropriate for their product.
- The effort required to produce a design rationale forces the designer to deliberate more carefully about design decisions. The process of deliberation can be assisted by the design rationale technique by suggesting how arguments justifying or discarding a particular design option are formed.

In the area of HCI, design rationale has been particularly important, again for several reasons:

- There is usually no single best design alternative. More often, the designer is faced with a set of trade-offs between different alternatives. For example, a graphical interface may involve a set of actions that the user can invoke by use of the mouse and the designer must decide whether to present each action as a 'button' on the screen, which is always visible, or hide all of the actions in a menu which must be explicitly invoked before an action can be chosen. The former option maximizes the operation visibility (see Chapter 7) but the latter option takes up less screen space. It would be up to the designer to determine which criterion for evaluating the options was more important and then communicating that information in a design rationale.
- Even if an optimal solution did exist for a given design decision, the space of alternatives is so vast that it is unlikely a designer would discover it. In this case, it is important that the designer indicates all alternatives that have been investigated. Then later on it can be determined if she has not considered the best solution or had thought about it and discarded it for some reason. In project management, this kind of accountability for design is good.
- The usability of an interactive system is very dependent on the context of its use. The flashiest graphical interface is of no use if the end-user does not have access to a high-quality graphics display or a pointing device. Capturing the context in

which a design decision is made will help later when new products are designed. If the context remains the same, then the old rationale can be adopted without revision. If the context has changed somehow, the old rationale can be re-examined to see if any rejected alternatives are now more favorable or if any new alternatives are now possible.

Lee and Lai [209] explain that various proponents of design rationale have different interpretations of what it actually is. We will make use of their classification to describe various design rationale techniques in this section. The first set of techniques concentrates on providing a historical record of design decisions and is very much tailored for use during actual design discussions. These techniques are referred to as process-oriented design rationale because they are meant to be integrated in the actual design process itself. The next category is not so concerned with historical or process-oriented information but rather with the structure of the space of all design alternatives, which can be reconstructed by post hoc consideration of the design activity. The structure-oriented approach does not capture historical information. Instead, it captures the complete story of the moment, as an analysis of the design space which has been considered so far. The final category of design rationale concentrates on capturing the claims about the psychology of the user that are implied by an interactive system and the tasks that are performed on them.

There are some issues that distinguish the various techniques in terms of their usability within design itself. We can use these issues to sketch an informal rationale for design rationale. One issue is the degree to which the technique impinges on the design process. Does the use of a particular design rationale technique alter the decision process, or does it just passively serve to document it? Another issue is the cost of using the technique, both in terms of creating the design rationale and in terms of accessing it once created. A related issue is the amount of computational power the design rationale provides and the level to which this is supported by automated tools. A design rationale for a complex system can be very large and the exploration of the design space changes over time. The kind of information stored in a given design rationale will affect how that vast amount of information can be effectively managed and browsed.

6.5.1 Process-oriented design rationale

Much of the work on design rationale is based on Rittel's *issue-based information system*, or *IBIS*, a style for representing design and planning dialog developed in the 1970s [308]. In IBIS (pronounced 'ibbiss'), a hierarchical structure to a design rationale is created. A root *issue* is identified which represents the main problem or question that the argument is addressing. Various *positions* are put forth as potential resolutions for the root issue, and these are depicted as descendants in the IBIS hierarchy directly connected to the root issue. Each position is then supported or refuted by *arguments*, which modify the relationship between issue and position. The hierarchy grows as secondary issues are raised which modify the root issue in some way. Each of these secondary issues is in turn expanded by positions and arguments, further sub-issues, and so on.

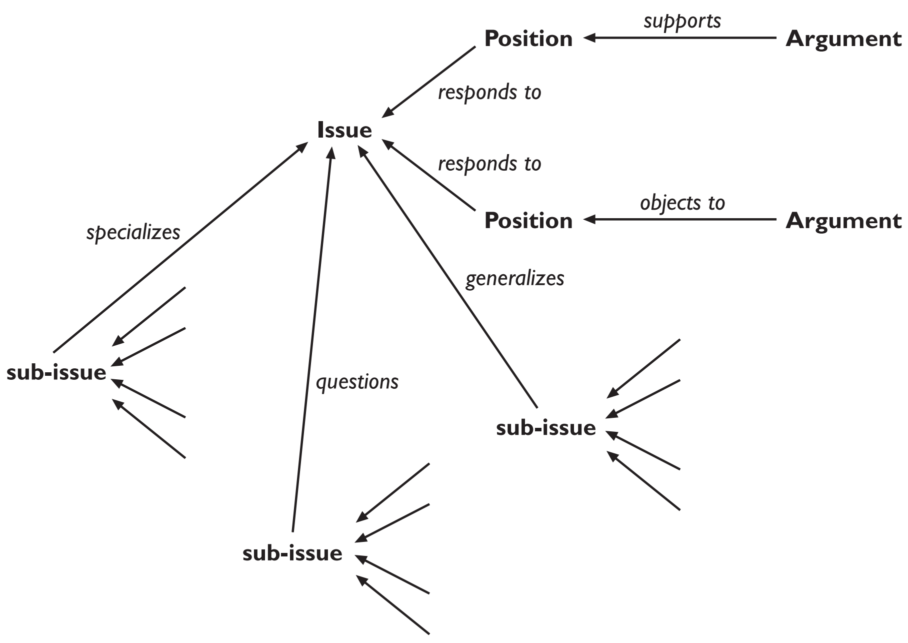


Figure 6.8 The structure of a gIBIS design rationale

A graphical version of IBIS has been defined by Conklin and Yakemovic [77], called *gIBIS* (pronounced 'gibbiss'), which makes the structure of the design rationale more apparent visually in the form of a directed graph which can be directly edited by the creator of the design rationale. Figure 6.8 gives a representation of the gIBIS vocabulary. Issues, positions and arguments are nodes in the graph and the connections between them are labeled to clarify the relationship between adjacent nodes. So, for example, an issue can suggest further sub-issues, or a position can respond to an issue or an argument can support a position. The gIBIS structure can be supported by a hypertext tool to allow a designer to create and browse various parts of the design rationale.

There have been other versions of the IBIS notation, both graphical and textual, besides gIBIS. Most versions retain the distinction between issues, positions and arguments. Some add further nodes, such as Potts and Bruns's [297] addition of design artifacts which represent the intermediate products of a design that lead to the final product and are associated with the various alternatives discussed in the design rationale. Some add a richer vocabulary to modify the relationships between the node elements, such as McCall's Procedural Hierarchy of Issues (PHI) [231], which expands the variety of inter-issue relationships. Interesting work at the University of Colorado has attempted to link PHI argumentation to computer-aided design (CAD) tools to allow critique of design (in their example, the design of a kitchen) as it occurs. When the CAD violates some known design rule, the designer is warned and can then browse a PHI argument to see the rationale for the design rule.

The use of IBIS and any of its descendants is process oriented, as we described above. It is intended for use during design meetings as a means of recording and structuring the issues deliberated and the decisions made. It is also intended to preserve the order of deliberation and decision making for a particular product, placing less stress on the generalization of design knowledge for use between different products. This can be contrasted with the structure-oriented technique discussed next.

6.5.2 Design space analysis

MacLean and colleagues [222] have proposed a more deliberative approach to design rationale which emphasizes a post hoc structuring of the space of design alternatives that have been considered in a design project. Their approach, embodied in the Questions, Options and Criteria (QOC) notation, is characterized as *design space analysis* (see Figure 6.9).

The design space is initially structured by a set of questions representing the major issues of the design. Since design space analysis is structure oriented, it is not so important that the questions recorded are the actual questions asked during design meetings. Rather, these questions represent an agreed characterization of the

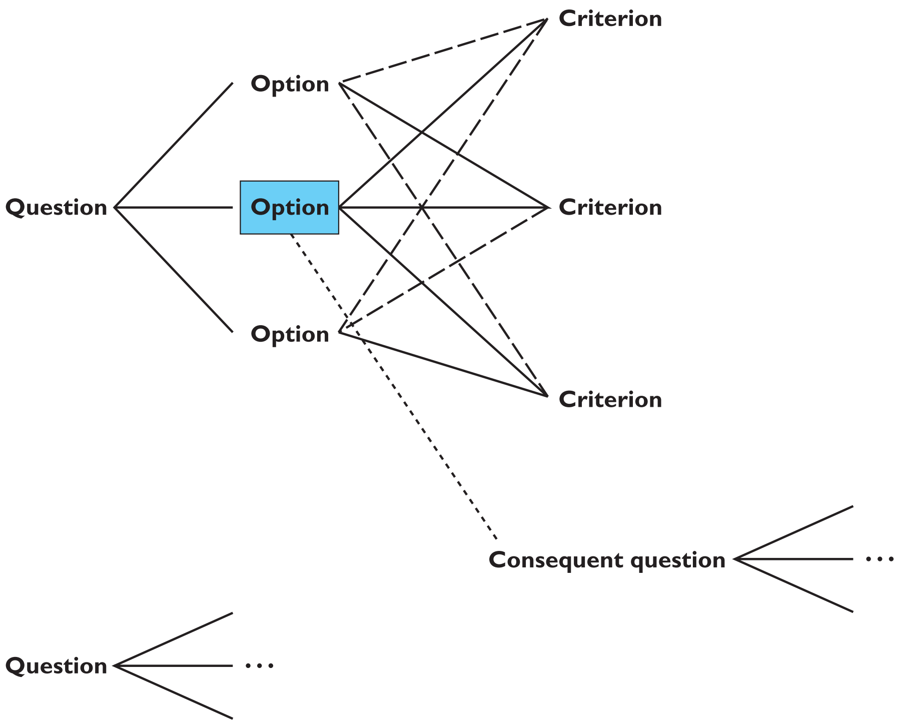


Figure 6.9 The QOC notation

issues raised based on reflection and understanding of the actual design activities. Questions in a design space analysis are therefore similar to issues in IBIS except in the way they are captured. Options provide alternative solutions to the question. They are assessed according to some criteria in order to determine the most favorable option. In Figure 6.9 an option which is favorably assessed in terms of a criterion is linked with a solid line, whereas negative links have a dashed line. The most favorable option is boxed in the diagram.

The key to an effective design space analysis using the QOC notation is deciding the right questions to use to structure the space and the correct criteria to judge the options. The initial questions raised must be sufficiently general that they cover a large enough portion of the possible design space, but specific enough that a range of options can be clearly identified. It can be difficult to decide the right set of criteria with which to assess the options. The QOC technique advocates the use of general criteria, like the usability principles we shall discuss in Chapter 7, which are expressed more explicitly in a given analysis. In the example of the action buttons versus the menu of actions described earlier, we could contextualize the general principle of operation visibility as the criterion that all possible actions are displayed at all times. It can be very difficult to decide from a design space analysis which option is most favorable. The positive and negative links in the QOC notation do not provide all of the context for a trade-off decision. There is no provision for indicating, for example, that one criterion is more important than any of the others and the most favorable option must be positively linked.

Another structure-oriented technique, called Decision Representation Language (DRL), developed by Lee and Lai, structures the design space in a similar fashion to QOC, though its language is somewhat larger and it has a formal semantics. The questions, options and criteria in DRL are given the names: decision problem, alternatives and goals. QOC assessments are represented in DRL by a more complex language for relating goals to alternatives. The sparse language in QOC used to assess an option relative to a criterion (positive or negative assessment only) is probably insufficient, but there is a trade-off involved in adopting a more complex vocabulary which may prove too difficult to use in practice. The advantage of the formal semantics of DRL is that the design rationale can be used as a computational mechanism to help manage the large volume of information. For example, DRL can track the dependencies between different decision problems, so that subsequent changes to the design rationale for one decision problem can be automatically propagated to other dependent problems.

Design space analysis directly addresses the claim that no design activity can hope to uncover all design possibilities, so the best we can hope to achieve is to document the small part of the design space that has been investigated. An advantage of the post hoc technique is that it can abstract away from the particulars of a design meeting and therefore represent the design knowledge in such a way that it can be of use in the design of other products. The major disadvantage is the increased overhead such an analysis warrants. More time must be taken away from the design activity to do this separate documentation task. When time is scarce, these kinds of overhead costs are the first to be trimmed.

6.5.3 Psychological design rationale

The final category of design rationale tries to make explicit the psychological claims of usability inherent in any interactive system in order better to suit a product for the tasks users have. This psychological design rationale has been introduced by Carroll and Rosson [62], and before we describe the application of the technique it is important to understand some of its theoretical background.

People use computers to accomplish some tasks in their particular work domain, as we have seen before. When designing a new interactive system, the designers take into account the tasks that users currently perform and any new ones that they may want to perform. This task identification serves as part of the requirements for the new system, and can be done through empirical observation of how people perform their work currently and presented through informal language or a more formal task analysis language (see Chapter 15). When the new system is implemented, or becomes an *artifact*, further observation reveals that in addition to the required tasks it was built to support, it also supports users in tasks that the designer never intended. Once designers understand these new tasks, and the associated problems that arise between them and the previously known tasks, the new task definitions can serve as requirements for future artifacts.

Carroll refers to this real-life phenomenon as the *task–artifact cycle*. He provides a good example of this cycle through the evolution of the electronic spreadsheet. When the first electronic spreadsheet, *VisiCalc*, was marketed in the late 1970s, it was presented simply as an automated means of supporting tabular calculation, a task commonly used in the accounting world. Within little over a decade of its introduction, the application of spreadsheets had far outstripped its original intent within accounting. Spreadsheets were being used for all kinds of financial analysis, 'what-if' simulations, report formatting and even as a general programming language paradigm! As the set of tasks expands, new spreadsheet products have flooded the marketplace trying to satisfy the growing customer base. Another good example of the task–artifact cycle in action is with word processing, which was originally introduced to provide more automated support for tasks previously achieved with a typewriter and now provides users with the ability to carry out various authoring tasks that they never dreamed possible with a conventional typewriter. And today, the tasks for the spreadsheet and the word processor are intermingled in the same artifact.

The purpose of psychological design rationale is to support this natural task–artifact cycle of design activity. The main emphasis is not to capture the designer's intention in building the artifact. Rather, psychological design rationale aims to make explicit the consequences of a design for the user, given an understanding of what tasks he intends to perform. Previously, these psychological consequences were left implicit in the design, though designers would make informal claims about their systems (for example, that it is more 'natural' for the user, or easier to learn).

The first step in the psychological design rationale is to identify the tasks that the proposed system will address and to characterize those tasks by questions that the user tries to answer in accomplishing them. For instance, Carroll gives an example of designing a system to help programmers learn the Smalltalk object-oriented programming language environment. The main task the system is to support is learning how Smalltalk works. In learning about the programming environment, the programmer will perform tasks that help her answer the questions:

- What can I do: that is, what are the possible operations or functions that this programming environment allows?
- How does it work: that is, what do the various functions do?
- How can I do this: that is, once I know a particular operation I want to perform, how do I go about programming it?

For each question, a set of *scenarios* of user–system behavior is suggested to support the user in addressing the question. For example, to address the question 'What can I do?', the designers can describe a scenario whereby the novice programmer is first confronted with the learning environment and sees that she can invoke some demo programs to investigate how Smalltalk programs work. The initial system can then be implemented to provide the functionality suggested by the scenarios (for example, some demos would be made accessible and obvious to the user/programmer from the very beginning). Once this system is running, observation of its use and some designer reflection is used to produce the actual psychological design rationale for that version of the system. This is where the psychological claims are made explicit. For example, there is an assumption that the programmer knows that what she can see on the screen relates to what she can do (if she sees the list of programs under a heading 'Demos', she can click on one program name to see the associated demo). The psychological claim of this demo system is that the user learns by doing, which is a good thing. However, there may also be negative aspects that are equally important to mention. The demo may not be very interactive, in which case the user clicks on it to initiate it and then just sits back and watches a graphic display, never really learning how the demo application is constructed in Smalltalk. These negative aspects can be used to modify later versions of the system to allow more interactive demos, which represent realistic, yet simple, applications, whose behavior and structure the programmer can appreciate.

By forcing the designer to document the psychological design rationale, it is hoped that she will become more aware of the natural evolution of user tasks and the artifact, taking advantage of how consequences of one design can be used to improve later designs.

Worked exercise *What is the distinction between a process-oriented and a structure-oriented design rationale technique? Would you classify psychological design rationale as process or structure oriented? Why?*

Answer The distinction between a process- and structure-oriented design rationale resides in what information the design rationale attempts to capture. Process-oriented design rationale is interested in recording an historically accurate description of a design team making some decision on a particular issue for the design. In this sense, process-oriented design rationale becomes an activity concurrent with the rest of the design process. Structure-oriented design rationale is less interested in preserving the historical evolution of the design. Rather, it is more interested in providing the conclusions of the design activity, so it can be done in a post hoc and reflective manner after the fact.

The purpose of psychological design rationale is to support the task–artifact cycle. Here, the tasks that the users perform are changed by the systems on which they perform the tasks. A psychological design rationale proceeds by having the designers of the system record what they believe are the tasks that the system should support and then building the system to support the tasks. The designers suggest scenarios for the tasks which will be used to observe new users of the system. Observations of the users provide the information needed for the actual design rationale of that version of the system. The consequences of the design's assumptions about the important tasks are then gauged against the actual use in an attempt to justify the design or suggest improvements.

Psychological design rationale is mainly a process-oriented approach. The activity of a claims analysis is precisely about capturing what the designers assumed about the system at one point in time and how those assumptions compared with actual use. Therefore, the history of the psychological design rationale is important. The discipline involved in performing a psychological design rationale requires designers to perform the claims analysis during the actual design activity, and not as post hoc reconstruction.

6.6 SUMMARY

In this chapter, we have shown how software engineering and the design process relate to interactive system design. The software engineering life cycle aims to structure design in order to increase the reliability of the design process. For interactive system design, this would equate to a reliable and reproducible means of designing predictably usable systems. Because of the special needs of interactive systems, it is essential to augment the standard life cycle in order to address issues of HCI.

Usability engineering encourages incorporating explicit usability goals within the design process, providing a means by which the product's usability can be judged. Iterative design practices admit that principled design of interactive systems alone cannot maximize product usability, so the designer must be able to evaluate early prototypes and rapidly correct features of the prototype which detract from the product usability.

The design process is composed of a series of decisions, which pare down the vast set of potential systems to the one that is actually delivered to the customer. Design rationale, in its many forms, is aimed at allowing the designer to manage the information about the decision-making process, in terms of when and why design decisions were made and what consequences those decisions had for the user in accomplishing his work.

Exercises

6.1 (a) How can design rationale benefit interface design and why might it be rejected by design teams?
(b) Explain QOC design rationale using an example to illustrate.

6.2 Imagine you have been asked to produce a prototype for the diary system discussed in the worked exercise in Section 7.2.3. What would be an appropriate prototyping approach to enable you to test the design using the usability metrics specified, and why?

Recommended reading

J. A. McDermid, editor, *The Software Engineer's Reference Book*, Butterworth–Heinemann, 1992.
A very good general reference book for all topics in software engineering. In particular, we refer you to Chapter 15 on software life cycles and Chapter 40 on prototyping.

I. Sommerville, *Software Engineering*, 6th edition, Addison-Wesley, 2000.
This textbook is one of the few texts in software engineering that specifically treats issues of interface design.

X. Faulkner, *Usability Engineering*, Macmillan, 2000.
An excellent and accessible introduction to usability engineering covering, amongst other things, user requirements capture and usability metrics.

J. Whiteside, J. Bennett and K. Holtzblatt, Usability engineering: our experience and evolution. In M. Helander, editor, *Handbook for Human–Computer Interaction*, North-Holland, 1988.
The seminal work on usability engineering. More recent work on usability engineering has also been published by Jakob Nielsen [260, 261].

J. M. Carroll and T. P. Moran, editors, *Design Rationale: Concepts, Techniques and Use*, Lawrence Erlbaum, 1996.
Expanded from a double special journal issue, this provides comprehensive coverage of relevant work in the field.

7 Design rules

OVERVIEW

- Designing for maximum usability is the goal of interactive systems design.
- Abstract principles offer a way of understanding usability in a more general sense, especially if we can express them within some coherent catalog.
- Design rules in the form of standards and guidelines provide direction for design, in both general and more concrete terms, in order to enhance the interactive properties of the system.
- The essential characteristics of good design are often summarized through ‘golden rules’ or heuristics.
- Design patterns provide a potentially generative approach to capturing and reusing design knowledge.

7.1 INTRODUCTION

One of the central problems that must be solved in a user-centered design process is how to provide designers with the ability to determine the usability consequences of their design decisions. We require *design rules*, which are rules a designer can follow in order to increase the usability of the eventual software product. We can classify these rules along two dimensions, based on the rule's authority and generality. By authority, we mean an indication of whether or not the rule must be followed in design or whether it is only suggested. By generality, we mean whether the rule can be applied to many design situations or whether it is focussed on a more limited application situation. Rules also vary in their level of abstraction, with some abstracting away from the detail of the design solution and others being quite specific. It is also important to determine the origins of a design rule. We will consider a number of different types of design rules. *Principles* are abstract design rules, with high generality and low authority. *Standards* are specific design rules, high in authority and limited in application, whereas *guidelines* tend to be lower in authority and more general in application.

Design rules for interactive systems can be supported by psychological, cognitive, ergonomic, sociological, economic or computational theory, which may or may not have roots in empirical evidence. Designers do not always have the relevant background in psychology, cognitive science, ergonomics, sociology, business or computer science necessary to understand the consequences of those theories in the instance of the design they are creating. The design rules are used to apply the theory in practice. Often a set of design rules will be in conflict with each other, meaning that strict adherence to all of them is impossible. The theory underlying the separate design rules can help the designer understand the trade-off for the design that would result in following or disregarding some of the rules. Usually, the more general a design rule is, the greater the likelihood that it will conflict with other rules and the greater the need for the designer to understand the theory behind it.

We can make another rough distinction between principles, standards and guidelines. Principles are derived from knowledge of the psychological, computational and sociological aspects of the problem domains and are largely independent of the technology; they depend to a much greater extent on a deeper understanding of the human element in the interaction. They can therefore be applied widely but are not so useful for specific design advice. Guidelines are less abstract and often more technology oriented, but as they are also general, it is important for a designer to know what theoretical evidence there is to support them. A designer will have less of a need to know the underlying theory for applying a standard. However, since standards carry a much higher level of authority, it is more important that the theory underlying them be correct or sound.

The previous chapter was about the process of design, and we need to consider when design rules can be of use within that process. Design rules are mechanisms for restricting the space of design options, preventing a designer from pursuing design options that would be likely to lead to an unusable system. Thus, design rules would be most effective if they could be adopted in the earliest stages of the life cycle, such as in requirements specification and architectural design, when the space of possible designs is still very large. We have already seen, for example, in Chapter 6, how abstract principles can be applied in usability engineering.

However, if the designer does not understand the assumptions underlying a design rule, it is quite possible that early application can prevent the best design choice. For example, a set of design rules might be specific to a particular hardware platform and inappropriate for other platforms (for example, color versus monochrome screens, one- versus two- or three-button mouse). Such bias in some design rules causes them to be applicable only in later stages of the life cycle.

We will first discuss abstract principles, then go on to consider in more depth some examples of standards and guidelines for user-centered design. Finally, we will consider some well-known heuristics or 'golden rules' which, it has been suggested, provide a succinct summary of the essence of good design. We end the chapter with a discussion of design patterns, a relatively new approach to capturing design knowledge in HCI.

7.2 PRINCIPLES TO SUPPORT USABILITY

The most abstract design rules are general principles, which can be applied to the design of an interactive system in order to promote its usability. In Chapter 4 we looked at the different paradigms that represent the development of interactive systems. Derivation of principles for interaction has usually arisen out of a need to explain why a paradigm is successful and when it might not be. Principles can provide the repeatability which paradigms in themselves cannot provide. In this section we present a collection of usability principles. Since it is too bold an objective to produce a comprehensive catalog of such principles, our emphasis will be on structuring the presentation of usability principles in such a way that the catalog can be easily extended as our knowledge increases.

The principles we present are first divided into three main categories:

Learnability – the ease with which new users can begin effective interaction and achieve maximal performance.

Flexibility – the multiplicity of ways in which the user and system exchange information.

Robustness – the level of support provided to the user in determining successful achievement and assessment of goals.

In the following, we will subdivide these main categories into more specific principles that support them. In most cases, we are able to situate these more specific principles within a single category, but we have made explicit those cases when a principle falls into two of the above categories.

Table 7.1 Summary of principles affecting learnability

Principle	Definition	Related principles
Predictability	Support for the user to determine the effect of future action based on past interaction history	Operation visibility
Synthesizability	Support for the user to assess the effect of past operations on the current state	Immediate/eventual honesty
Familiarity	The extent to which a user's knowledge and experience in other real-world or computer-based domains can be applied when interacting with a new system	Guessability, affordance
Generalizability	Support for the user to extend knowledge of specific interaction within and across applications to other similar situations	–
Consistency	Likeness in input–output behavior arising from similar situations or similar task objectives	–

7.2.1 Learnability

Learnability concerns the features of the interactive system that allow novice users to understand how to use it initially and then how to attain a maximal level of performance. Table 7.1 contains a summary of the specific principles that support learnability, which we will describe below.

Predictability

Except when interacting with some video games, a user does not take very well to surprises. Predictability of an interactive system means that the user's knowledge of the interaction history is sufficient to determine the result of his future interaction with it. There are many degrees to which predictability can be satisfied. The knowledge can be restricted to the presently perceivable information, so that the user need not remember anything other than what is currently observable. The knowledge requirement can be increased to the limit where the user is actually forced to remember what every previous keystroke was and what every previous screen display contained (and the order of each!) in order to determine the consequences of the next input action.

Predictability of an interactive system is distinguished from deterministic behavior of the computer system alone. Most computer systems are ultimately deterministic machines, so that given the state at any one point in time and the operation which is to be performed at that time, there is only one possible state that can result. Predictability is a user-centered concept; it is deterministic behavior from the perspective of the user. It is not enough for the behavior of the computer system to be determined completely from its state, as the user must be able to take advantage of the determinism.

For example, a common mathematical puzzle would be to present you with a sequence of three or more numbers and ask you what would be the next number in the sequence. The assumption in this puzzle (and one that can often be incorrect) is that there is a unique function or algorithm that produces the entire sequence of numbers and it is up you to figure it out. We know the function, but all you know are the results it provides from the first three calculations. The function is certainly deterministic; the test for you is a test of its predictability given the first three numbers in the sequence.

As another, possibly more pertinent, example, imagine you have created a complex picture using a mouse-driven graphical drawing package. You leave the picture for a few days and then go back to change it around a bit. You are allowed to select certain objects for editing by positioning the mouse over the object and clicking a mouse button to highlight it. Can you tell what the set of selectable objects is? Can you determine which area of the screen belongs to which of these objects, especially if some objects overlap? Does the visual image on the screen indicate what objects form a compound object that can only be selected as a group? Predictability of selection in this example depends on how much of the history of the creation of the visual image is necessary in order for you to determine what happens when you click on the mouse button.

This notion of predictability deals with the user's ability to determine the effect of operations on the system. Another form of predictability has to do with the user's ability to know which operations can be performed. *Operation visibility* refers to how the user is shown the availability of operations that can be performed next. If an operation can be performed, then there may be some perceivable indication of this to the user. This principle supports the superiority in humans of recognition over recall. Without it, the user will have to remember when he can perform the operation and when he cannot. Likewise, the user should understand from the interface if an operation he might like to invoke cannot be performed.

Synthesizability

Predictability focusses on the user's ability to determine the effect of future interactions. This assumes that the user has some mental model (see Chapter 1) of how the system behaves. Predictability says nothing about the way the user forms a model of the system's behavior. In building up some sort of predictive model of the system's behavior, it is important for the user to assess the consequences of previous interactions in order to formulate a model of the behavior of the system. Synthesis, therefore, is the ability of the user to assess the effect of past operations on the current state.

When an operation changes some aspect of the internal state, it is important that the change is seen by the user. The principle of *honesty* relates to the ability of the user interface to provide an observable and informative account of such change. In the best of circumstances, this notification can come *immediately*, requiring no further interaction initiated by the user. At the very least, the notification should appear *eventually*, after explicit user directives to make the change observable. A good example of the distinction between immediacy and eventuality can be seen in the comparison between command language interfaces and visual desktop interfaces for a file management system. You have moved a file from one directory to another. The principle of honesty implies that after moving the file to its new location in the file system you are then able to determine its new whereabouts. In a command language system, you would typically have to remember the destination directory and then ask to see the contents of that directory in order to verify that the file has been moved (in fact, you would also have to check that the file is no longer in its original directory to determine that it has been moved and not copied). In a visual desktop interface, a visual representation (or icon) of the file is dragged from its original directory and placed in its destination directory where it remains visible (assuming the destination folder is selected to reveal its contents). In this case, the user need not expend any more effort to assess the result of the move operation. The visual desktop is immediately honest.

The problem with eventual honesty is that the user must know to look for the change. In a situation in which the user is learning a new interactive system, it is likely that he will not know to look for change. In earlier versions of the Apple Macintosh Finder, performing the operation to create a new folder in another folder did not necessarily result in that new folder's icon being visible in the original folder. New users (and even some experienced users) would often think that they had not issued the new folder operations correctly and would ask for another new folder (and another, and another, . . .). They would not know to search through the entire open folder for the latest addition. Then several minutes (hours, days) later, they would notice that there were a number of empty and untitled folders lying around. The eventual (accidental) discovery of the change brought about by the new folder operation was then difficult to associate to that operation. Fortunately, this problem was addressed in Version 7 of the Finder.

As another example of the benefit of immediate over eventual honesty, let us examine a typical global search and replace function in a word processor. Imagine you have noticed in the past a tendency to repeat words in a document (for example, you type 'the the' without noticing the error). In an attempt to automate your proofreading, you decide to replace globally all occurrences of 'the the' with 'the'. The typical global search and replace function performs this substitution without revealing the changes made to you. Suddenly, a careless typing error is transformed into unacceptable grammar as the sentence

> We will prove the theorem holds as a corollary of the following lemma.

is transformed to

> We will prove theorem holds as a corollary of the following lemma.

Familiarity

New users of a system bring with them a wealth of experience across a wide number of application domains. This experience is obtained both through interaction in the real world and through interaction with other computer systems. For a new user, the familiarity of an interactive system measures the correlation between the user's existing knowledge and the knowledge required for effective interaction. For example, when word processors were originally introduced the analogy between the word processor and a typewriter was intended to make the new technology more immediately accessible to those who had little experience with the former but a lot of experience with the latter. Familiarity has to do with a user's first impression of the system. In this case, we are interested in how the system is first perceived and whether the user can determine how to initiate any interaction. An advantage of a metaphor, such as the typewriter metaphor for word processing described above, is precisely captured by familiarity. Jordan et al. refer to this familiarity as the *guessability* of the system [196].

Some psychologists argue that there are intrinsic properties, or *affordances*, of any visual object that suggest to us how they can be manipulated (see also Chapter 5, Section 5.7.2). The appearance of the object stimulates a familiarity with its behavior. For example, the shape of a door handle can suggest how it should be manipulated to open a door, and a key on a keyboard suggests to us that it can be pushed. In the design of a graphical user interface, it is implied that a soft button used in a form's interface suggests it should be pushed (though it does not suggest how it is to be pushed via the mouse). Effective use of the affordances that exist for interface objects can enhance the familiarity of the interactive system.

Generalizability

Users often try to extend their knowledge of specific interaction behavior to situations that are similar but previously unencountered. The generalizability of an interactive system supports this activity, leading to a more complete predictive model of the system for the user. We can apply generalization to situations in which the user wants to apply knowledge that helps achieve one particular goal to another situation where the goal is in some way similar. Generalizability can be seen as a form of consistency.

Generalization can occur within a single application or across a variety of applications. For example, in a graphical drawing package that draws a circle as a constrained form of ellipse, we would want the user to generalize that a square can be drawn as a constrained rectangle. A good example of generalizability across a variety of applications can be seen in multi-windowing systems that attempt to provide cut/paste/copy operations to all applications in the same way (with varying degrees of success). Generalizability within an application can be maximized by any conscientious designer. One of the main advantages of standards and programming style guides, which we will discuss in Sections 7.3 and 7.4, is that they increase generalizability across a wide variety of applications within the same environment.

Consistency

Consistency relates to the likeness in behavior arising from similar situations or similar task objectives. Consistency is probably the most widely mentioned principle in the literature on user interface design. 'Be consistent!' we are constantly urged. The user relies on a consistent interface. However, the difficulty of dealing with consistency is that it can take many forms. Consistency is not a single property of an interactive system that is either satisfied or not satisfied. Instead, consistency must be applied relative to something. Thus we have consistency in command naming, or consistency in command/argument invocation.

Another consequence of consistency having to be defined with respect to some other feature of the interaction is that many other principles can be 'reduced' to qualified instances of consistency. Hence, familiarity can be considered as consistency with respect to past real-world experience, and generalizability as consistency with respect to experience with the same system or set of applications on the same platform. Because of this pervasive quality of consistency, it might be argued that consistency should be a separate category of usability principles, on the same level as learnability, flexibility and robustness. Rather than do that, we will discuss different ways in which consistency can be manifested.

Consistency can be expressed in terms of the form of input expressions or output responses with respect to the meaning of actions in some conceptual model of the system. For example, before the introduction of explicit arrow keys, some word processors used the relative position of keys on the keyboard to indicate directionality for operations (for example, to move one character to the left, right, up or down). The conceptual model for display-based editing is a two-dimensional plane, so the user would think of certain classes of operations in terms of movements up, down, left or right in the plane of the display. Operations that required directional information, such as moving within the text or deleting some unit of text, could be articulated by using some set of keys on the keyboard that form a pattern consistent with up, down, left and right (for example, the keys e, x, s and d, respectively). For output responses, a good example of consistency can be found in a warnings system for an aircraft. Warnings to the pilot are classified into three categories, depending on whether the situation with the aircraft requires immediate recovery action, eventual but not immediate action, or no action at all (advisory) on the part of the crew. These warnings are signalled to the crew by means of a centralized warnings panel in which the categories are consistently color coded (red for immediate, amber for eventual and green for advisory).

Grudin has argued that because of the relative nature of consistency it can be a dangerous principle to follow [160]. A good example he gives is the development and evolution of the standard typewriter keyboard. When keyboards for typewriters were first made, the designers laid out the keys in alphabetical order. Then it was discovered that such an arrangement of keys was both inefficient from the machine's perspective (adjacent typewriter keys pressed in succession caused jams in the mechanism, so the likelihood of this occurrence had to be designed out) and tiring for the typist (a touch-typist would not have equal stress distributed over all fingers). The resulting QWERTY and DVORAK keyboards have since been adopted to combat the problems of the 'consistent' keyboard layout.[1]

1 See Chapter 2 for a discussion of different keyboards.

Table 7.2 Summary of principles affecting flexibility

Principle	Definition	Related principles
Dialog initiative	Allowing the user freedom from artificial constraints on the input dialog imposed by the system	System/user pre-emptiveness
Multi-threading	Ability of the system to support user interaction pertaining to more than one task at a time	Concurrent vs. interleaving, multi-modality
Task migratability	The ability to pass control for the execution of a given task so that it becomes either internalized by the user or the system or shared between them	–
Substitutivity	Allowing equivalent values of input and output to be arbitrarily substituted for each other	Representation multiplicity, equal opportunity
Customizability	Modifiability of the user interface by the user or the system	Adaptivity, adaptability

7.2.2 Flexibility

Flexibility refers to the multiplicity of ways in which the end-user and the system exchange information. We identify several principles that contribute to the flexibility of interaction, and these are summarized in Table 7.2.

Dialog initiative

When considering the interaction between user and system as a dialog between partners (see Chapter 16), it is important to consider which partner has the initiative in the conversation. The system can initiate all dialog, in which case the user simply responds to requests for information. We call this type of dialog *system pre-emptive*. For example, a modal dialog box prohibits the user from interacting with the system in any way that does not direct input to the box. Alternatively, the user may be entirely free to initiate any action towards the system, in which case the dialog is *user pre-emptive*. The system may control the dialog to the extent that it prohibits the user from initiating any other desired communication concerning the current task or some other task the user would like to perform. From the user's perspective, a system-driven interaction hinders flexibility whereas a user-driven interaction favours it.

In general, we want to maximize the user's ability to pre-empt the system and minimize the system's ability to pre-empt the user. Although a system pre-emptive dialog is not desirable in general, some situations may require it. In a cooperative editor (in which two people edit a document at the same time) it would be impolite for you to erase a paragraph of text that your partner is currently editing. For safety reasons, it may be necessary to prohibit the user from the 'freedom' to do potentially serious damage. A pilot about to land an aircraft in which the flaps have asymmetrically failed in their extended position[2] should not be allowed to abort the landing, as this failure will almost certainly result in a catastrophic accident.

On the other hand, a completely user pre-emptive dialog allows the user to offer any input action at any time for maximum flexibility. This is not an entirely desirable situation, since it increases the likelihood that the user will lose track of the tasks that have been initiated and not yet completed. However, if the designers have a good understanding of the sets of tasks the user is likely to perform with a system and how those tasks are related, they can minimize the likelihood that the user will be prevented from initiating some task at a time when he wishes to do so.

Multi-threading

A thread of a dialog is a coherent subset of that dialog. In the user–system dialog, we can consider a thread to be that part of the dialog that relates to a given user task. *Multi-threading* of the user–system dialog allows for interaction to support more than one task at a time. *Concurrent* multi-threading allows simultaneous communication of information pertaining to separate tasks. *Interleaved* multi-threading permits a temporal overlap between separate tasks, but stipulates that at any given instant the dialog is restricted to a single task.

Multi-modality of a dialog is related to multi-threading. Coutaz has characterized two dimensions of multi-modal systems [80]. First, we can consider how the separate modalities (or channels of communication) are combined to form a single input or output expression. Multiple channels may be available, but any one expression may be restricted to just one channel (keyboard or audio, for example). As an example, to open a window the user can choose between a double click on an icon, a keyboard shortcut, or saying 'open window'. Alternatively, a single expression can be formed by a mixing of channels. Examples of such fused modality are error warnings, which usually contain a textual message accompanied by an audible beep. On the input side, we could consider chord sequences of input with a keyboard and mouse (pressing the shift key while a mouse button is pressed, or saying 'drop' as you drag a file over the trash icon). We can also characterize a multi-modality dialog depending on whether it allows concurrent or interleaved use of multiple modes.

A windowing system naturally supports a multi-threaded dialog that is interleaved amongst a number of overlapping tasks. Each window can represent a different task, for example text editing in one window, file management in another, a telephone directory in another and electronic mail in yet another. A multi-modal dialog can allow for concurrent multi-threading. A very simple example can occur in the

2 Flaps increase the surface area and curvature of the aircraft's wing, providing the extra lift necessary for, among other things, a smooth touchdown. An asymmetric failure results in extreme instability and the aircraft will not fly level.

windowing system with an audible bell. You are editing a program when a beep indicates that a new electronic mail message has arrived. Even though at the level of the system the audible beep has been interleaved with your requests from the keyboard to perform edits, the overlap between the editing task and the mail message from your perspective is simultaneous.

Task migratability

Task migratability concerns the transfer of control for execution of tasks between system and user. It should be possible for the user or system to pass the control of a task over to the other or promote the task from a completely internalized one to a shared and cooperative venture. Hence, a task that is internal to one can become internal to the other or shared between the two partners.

Spell-checking a paper is a good example of the need for task migratability. Equipped with a dictionary, you are perfectly able to check your spelling by reading through the entire paper and correcting mistakes as you spot them. This mundane task is perfectly suited to automation, as the computer can check words against its own list of acceptable spellings. It is not desirable, however, to leave this task completely to the discretion of the computer, as most computerized dictionaries do not handle proper names correctly, nor can they distinguish between correct and unintentional duplications of words. In those cases, the task is handed over to the user. The spell-check is best performed in such a cooperative way.

In safety-critical applications, task migratability can decrease the likelihood of an accident. For example, on the flight deck of an aircraft, there are so many control tasks that must be performed that a pilot would be overwhelmed if he had to perform them all. Therefore, mundane control of the aircraft's position within its flight envelope is greatly automated. However, in the event of an emergency, it must be possible to transfer flying controls easily and seamlessly from the system to the pilot.

Substitutivity

Substitutivity requires that equivalent values can be substituted for each other. For example, in considering the form of an input expression to determine the margin for a letter, you may want to enter the value in either inches or centimeters. You may also want to input the value explicitly (say 1.5 inches) or you may want to enter a calculation which produces the right input value (you know the width of the text is 6.5 inches and the width of the paper is 8.5 inches and you want the left margin to be twice as large as the right margin, so you enter $\frac{2}{3}$ (8.5 – 6.5) inches). This input substitutivity contributes towards flexibility by allowing the user to choose whichever form best suits the needs of the moment. By avoiding unnecessary calculations in the user's head, substitutivity can minimize user errors and cognitive effort.

We can also consider substitutivity with respect to output, or the system's rendering of state information. *Representation multiplicity* illustrates flexibility for state rendering. For example, the temperature of a physical object over a period of time can be presented as a digital thermometer if the actual numerical value is important or as a graph if it is only important to notice trends. It might even be desirable to make these representations simultaneously available to the user. Each representation provides a perspective on the internal state of the system. At a given time, the user is free to consider the representations that are most suitable for the current task.

Equal opportunity blurs the distinction between input and output at the interface. The user has the choice of what is input and what is output; in addition, output can be reused as input. Thimbleby describes this principle as, 'If you can see it, you can use it!' It is a common belief that input and output are separate. Many have stressed the significance of the link between input and output. Equal opportunity pushes that view to the extreme. For example, in spreadsheet programs, the user fills in some cells and the system automatically determines the values attributed to some other cells. Conversely, if the user enters values for those other cells, the system would compute the values for the first ones. In this example, it is not clear which cells are the inputs and which are the outputs. Furthermore, this distinction might not be clear or useful to the user. In a drawing package, the user may draw a line by direct manipulation and the system would compute the length of the line; or conversely, the user may specify the line coordinates and the system would draw the line. Both means of manipulating the line are equally important and must be made equally available. Note that equal opportunity implies that the system is not pre-emptive towards the user.

Customizability

Customizability is the modifiability of the user interface by the user or the system. From the system side, we are not concerned with modifications that would be attended to by a programmer actually changing the system and its interface during system maintenance. Rather, we are concerned with the automatic modification that the system would make based on its knowledge of the user. We distinguish between the user-initiated and system-initiated modification, referring to the former as *adaptability* and the latter as *adaptivity*.

Adaptability refers to the user's ability to adjust the form of input and output. This customization could be very limited, with the user only allowed to adjust the position of soft buttons on the screen or redefine command names. This type of modifiability, which is restricted to the surface of the interface, is referred to as lexical customization. The overall structure of the interaction is kept unchanged. The power given to the user can be increased by allowing the definition of macros to speed up the articulation of certain common tasks. In the extreme, the interface can provide the user with programming language capabilities, such as the UNIX shell or the script language Hypertalk in HyperCard. Thimbleby points out that in these cases it would be suitable to apply well-known principles of programming languages to the user's interface programming language.

Adaptivity is automatic customization of the user interface by the system. Decisions for adaptation can be based on user expertise or observed repetition of

Table 7.3 Summary of principles affecting robustness

Principle	Definition	Related principles
Observability	Ability of the user to evaluate the internal state of the system from its perceivable representation	Browsability, static/dynamic defaults, reachability, persistence, operation visibility
Recoverability	Ability of the user to take corrective action once an error has been recognized	Reachability, forward/backward recovery, commensurate effort
Responsiveness	How the user perceives the rate of communication with the system	Stability
Task conformance	The degree to which the system services support all of the tasks the user wishes to perform and in the way that the user understands them	Task completeness, task adequacy

certain task sequences. The distinction between adaptivity and adaptability is that the user plays an explicit role in adaptability, whereas his role in an adaptive interface is more implicit. A system can be trained to recognize the behavior of an expert or novice and accordingly adjust its dialog control or help system automatically to match the needs of the current user. This is in contrast with a system that would require the user to classify himself as novice or expert at the beginning of a session. We discuss adaptive systems further in the context of user support in Chapter 11. Automatic macro construction is a form of programming by example, combining adaptability with adaptivity in a simple and useful way. Repetitive tasks can be detected by observing user behavior and macros can be automatically (or with user consent) constructed from this observation to perform repetitive tasks automatically.

7.2.3 Robustness

In a work or task domain, a user is engaged with a computer in order to achieve some set of goals. The robustness of that interaction covers features that support the successful achievement and assessment of the goals. Here, we describe principles that support robustness. A summary of these principles is presented in Table 7.3.

Observability

Observability allows the user to evaluate the internal state of the system by means of its perceivable representation at the interface. As we described in Chapter 3, evaluation allows the user to compare the current observed state with his intention within the task–action plan, possibly leading to a plan revision. Observability can be discussed through five other principles: browsability, defaults, reachability, persistence and operation visibility. Operation visibility was covered in Section 7.2.1 in relation to predictability. The remaining four are discussed next.

Browsability allows the user to explore the current internal state of the system via the limited view provided at the interface. Usually the complexity of the domain does not allow the interface to show all of the relevant domain concepts at once. Indeed, this is one reason why the notion of task is used, in order to constrain the domain information needed at one time to a subset connected with the user's current activity. While you may not be able to view an entire document's contents, you may be able to see all of an outline view of the document, if you are only interested in its overall structure. Even with a restriction of concepts relevant to the current task, it is probable that all of the information a user needs to continue work on that task is not immediately perceivable. Or perhaps the user is engaged in a multi-threaded dialog covering several tasks. There needs to be a way for the user to investigate, or browse, the internal state. This browsing itself should not have any side-effects on that state; that is, the browsing commands should be passive with respect to the domain-specific parts of the internal state.

The availability of *defaults* can assist the user by passive recall (for example, a suggested response to a question can be recognized as correct instead of recalled). It also reduces the number of physical actions necessary to input a value. Thus, providing default values is a kind of error prevention mechanism. There are two kinds of default values: static and dynamic. Static defaults do not evolve with the session. They are either defined within the system or acquired at initialization. On the other hand, dynamic defaults evolve during the session. They are computed by the system from previous user inputs; the system is then adapting default values.

Reachability refers to the possibility of navigation through the observable system states. There are various levels of reachability that can be given precise mathematical definitions (see Chapter 17), but the main notion is whether the user can navigate from any given state to any other state. Reachability in an interactive system affects the recoverability of the system, as we will discuss later. In addition, different levels of reachability can reflect the amount of flexibility in the system as well, though we did not make that explicit in the discussion on flexibility.

Persistence deals with the duration of the effect of a communication act and the ability of the user to make use of that effect. The effect of vocal communication does not persist except in the memory of the receiver. Visual communication, on the other hand, can remain as an object which the user can subsequently manipulate long after the act of presentation. If you are informed of a new email message by a beep at your terminal, you may know at that moment and for a short while later that you have received a new message. If you do not attend to that message immediately, you may forget about it. If, however, some persistent visual information informs you of the incoming message (say, the flag goes up on your electronic mailbox), then that will serve as a reminder that an unread message remains long after its initial receipt.[3]

3 Chapter 19 discusses notification mechanisms for email in more detail.

Recoverability

Users make mistakes from which they want to recover. Recoverability is the ability to reach a desired goal after recognition of some error in a previous interaction. There are two directions in which recovery can occur, forward or backward. *Forward error recovery* involves the acceptance of the current state and negotiation from that state towards the desired state. Forward error recovery may be the only possibility for recovery if the effects of interaction are not revocable (for example, in building a house of cards, you might sneeze whilst placing a card on the seventh level, but you cannot undo the effect of your misfortune except by rebuilding). *Backward error recovery* is an attempt to undo the effects of previous interaction in order to return to a prior state before proceeding. In a text editor, a mistyped keystroke might wipe out a large section of text which you would want to retrieve by an equally simple undo button.

Recovery can be initiated by the system or by the user. When performed by the system, recoverability is connected to the notions of fault tolerance, safety, reliability and dependability, all topics covered in software engineering. However, in software engineering this recoverability is considered only with respect to system functionality; it is not tied to user intent. When recovery is initiated by the user, it is important that it determines the intent of the user's recovery actions; that is, whether he desires forward (negotiation) or backward (using undo/redo actions) corrective action.

Recoverability is linked to reachability because we want to avoid blocking the user from getting to a desired state from some other undesired state (going down a blind alley).

In addition to providing the ability to recover, the procedure for recovery should reflect the work being done (or undone, as the case may be). The principle of *commensurate effort* states that if it is difficult to undo a given effect on the state, then it should have been difficult to do in the first place. Conversely, easily undone actions should be easily doable. For example, if it is difficult to recover files which have been deleted in an operating system, then it should be difficult to remove them, or at least it should require more effort by the user to delete the file than to, say, rename it.

Responsiveness

Responsiveness measures the rate of communication between the system and the user. Response time is generally defined as the duration of time needed by the system to express state changes to the user. In general, short durations and instantaneous response times are desirable. Instantaneous means that the user perceives system reactions as immediate. But even in situations in which an instantaneous response cannot be obtained, there must be some indication to the user that the system has received the request for action and is working on a response.

As significant as absolute response time is response time *stability*. Response time stability covers the invariance of the duration for identical or similar computational resources. For example, pull-down menus are expected to pop up instantaneously as soon as a mouse button is pressed. Variations in response time will impede anticipation exploited by motor skill.

Task conformance

Since the purpose of an interactive system is to allow a user to perform various tasks in achieving certain goals within a specific application domain, we can ask whether the system supports all of the tasks of interest and whether it supports these as the user wants. *Task completeness* addresses the coverage issue and *task adequacy* addresses the user's understanding of the tasks.

It is not sufficient that the computer system fully implements some set of computational services that were identified at early specification stages. It is essential that the system allows the user to achieve any of the desired tasks in a particular work domain as identified by a task analysis that precedes system specification (see Chapter 15 for a more complete discussion of task analysis techniques). Task completeness refers to the level to which the system services can be mapped onto all of the user tasks. However, it is quite possible that the provision of a new computer-based tool will suggest to a user some tasks that were not even conceivable before the tool. Therefore, it is also desirable that the system services be suitably general so that the user can define new tasks.

Discussion of task conformance has its roots in an attempt to understand the success of direct manipulation interfaces. We can view the direct manipulation interface as a separate world from that inside the system. Task completeness covers only one part of the conformance. This separate world is understood and operated upon by the user. With the intuition of the Hutchins, Hollan and Norman model-world metaphor discussed in Chapter 4, we require that the task, as represented by the world of the interface, matches the task as understood by the user and supported by the system. If the model-world metaphor satisfies the principle of task adequacy, then the user will be directly on his task plan, minimizing the effort required in the articulation and observation translations discussed in the interaction framework of Chapter 3.

Worked exercise *Look at some of the principles outlined in this section, and use one or two to provide a usability specification (see Chapter 6, Section 6.3) for an electronic meetings diary or calendar. First identify some of the tasks that would be performed by a user trying to keep track of future meetings, and then complete the usability specification assuming that the electronic system will be replacing a paper-based system. What assumptions do you have to make about the user and the electronic diary in order to create a reasonable usability specification?*

Answer This exercise could be easily extended to a small project which would involve the design of such an electronic diary or calendar. The purpose of this smaller usability engineering exercise is to show how usability goals can be formulated early on to drive the design activity. We will select two of the usability principles from this chapter, which will serve as attributes for separate usability specifications.

In the first example, we will consider the interaction principle of guessability, which concerns how easy it is for new users to perform tasks initially. The measuring concept will be how long it takes a new user, without any instruction on the new system, to enter his first appointment in the diary. A sample usability specification is given below.

Attribute:	Guessability
Measuring concept:	Ease of first use of system without training
Measuring method:	Time to create first entry in diary
Now level:	30 seconds on paper-based system
Worst case:	1 minute
Planned level:	45 seconds
Best case:	30 seconds (equivalent to now)

The values in this usability specification might seem a little surprising at first, since we are saying that the best case is only equivalent to the currently achievable now level. The point in this example is that the new system is replacing a very familiar paper and pencil system which requires very little training. The objective of this system is not so much to improve guessability but to preserve it. Earlier, we discussed that the worst case level should not usually be worse than the now level, but we are hoping for this product to improve overall functionality of the system. The user will be able to do more things with the electronic diary than he could with the conventional system. As a result, we worry less about improving its guessability. Perhaps we could have been more ambitious in setting the best case value by considering the potential for voice input or other exotic input techniques that would make entry faster than writing.

As another example, we want to support the task migratability of the system. A frequent sort of task for a diary is to schedule weekly meetings. The conventional system would require the user to make an explicit entry for the meeting each week – the task of the scheduling is the responsibility of the user. In the new system, we want to allow the user to push the responsibility of scheduling over to the system, so that the user need only indicate the desire to have a meeting scheduled for a certain time each week and the system will take care of entering the meeting at all of the appropriate times. The task of scheduling has thus migrated over to the system. The usability specification for this example follows.

Attribute:	Task migratability
Measuring concept:	Scheduling a weekly meeting
Measuring method:	Time it takes to enter a weekly meeting appointment
Now level:	(Time to schedule one appointment) × (Number of weeks)
Worst case:	Time to schedule two appointments
Planned level:	1.5 × (Time to schedule one appointment)
Best case:	Time to schedule one appointment

In this specification, we have indicated that the now level is equivalent to the time it takes to schedule each appointment separately. The worst, planned and best case levels are all targeted at some proportion of the time it takes to schedule just a single appointment – a dramatic improvement. The difference between the worst, planned and best case levels is the amount of overhead it will take to indicate that a single appointment is to be considered an example to be repeated at the weekly level.

What are the assumptions we have to make in order to arrive at such a usability specification? One of the problems with usability specifications, discussed earlier, is that they sometimes require quite specific information about the design. For example, had we set one of our measuring methods to count keystrokes or mouse clicks, we would have had to start making assumptions about the method of interaction that the system would allow. Had we tried to set a usability specification concerning the browsing of the diary, we would have had to start making assumptions about the layout of the calendar (monthly, weekly, daily) in order to make our estimates specific enough to measure. In the examples we have provided above, we have tried to stay as abstract as possible, so that the usability specifications could be of use as early in the design life cycle as possible. A consequence of this abstractness, particularly evident in the second example, is that we run the risk in the usability specification of setting goals that may be completely unrealistic, though well intentioned. If the usability specification were to be used as a contract with the customer, such speculation could spell real trouble for the designer.

7.3 STANDARDS

Standards for interactive system design are usually set by national or international bodies to ensure compliance with a set of design rules by a large community. Standards can apply specifically to either the hardware or the software used to build the interactive system. Smith [324] points out the differing characteristics between hardware and software, which affect the utility of design standards applied to them:

Underlying theory Standards for hardware are based on an understanding of physiology or ergonomics/human factors, the results of which are relatively well known, fixed and readily adaptable to design of the hardware. On the other hand, software standards are based on theories from psychology or cognitive science, which are less well formed, still evolving and not very easy to interpret in the language of software design. Consequently, standards for hardware can directly relate to a hardware specification and still reflect the underlying theory, whereas software standards would have to be more vaguely worded.

Change Hardware is more difficult and expensive to change than software, which is usually designed to be very flexible. Consequently, requirements changes for hardware do not occur as frequently as for software. Since standards are also relatively stable, they are more suitable for hardware than software.

Historically, for these reasons, a given standards institution, such as the British Standards Institution (BSI) or the International Organization for Standardization (ISO) or a national military agency, has had standards for hardware in place before any for software. For example, the UK Ministry of Defence has published an Interim Defence Standard 00–25 on *Human Factors for Designers of Equipment*, produced in 12 parts:

Part 1	Introduction
Part 2	Body Size
Part 3	Body Strength and Stamina
Part 4	Workplace Design
Part 5	Stresses and Hazards
Part 6	Vision and Lighting

Part 7	Visual Displays
Part 8	Auditory Information
Part 9	Voice Communication
Part 10	Controls
Part 11	Design for Maintainability
Part 12	Systems

Only the last of these is concerned with the software design process. The international standard ISO 9241, entitled *Ergonomic Requirements for Office Work with Visual Display Terminals (VDT)s*, has 17 parts. Seven of these are concerned with hardware issues – requirements for visual display, keyboard layout, workstation layout, environment, display with reflections, display colors and non-keyboard input devices. Seven parts are devoted to software issues – general dialog principles, menu dialogs, presentation of information, user guidance, command dialogs, direct manipulation dialogs and form-filling dialogs. However, standards covering software issues are now being produced, for example, the draft standard ISO 14915 covers software ergonomics for multimedia user interfaces.

Figure 7.1 provides examples of the language of standards for displays. Note the increasing generality and vagueness of the language as we progress from the hardware issues in a UK defence standard for pilot cockpit controls and instrumentation through a German standard for user interface design of display workstations to a US military standard for display contents.

11.3 *Arrangement of displays*
11.3.1 Vertical Grouping. The engine display parameters shall be arranged so that the primary or most important display for a particular engine and airplane (thrust, torque, RPM, etc.) be located at the top of the display group if a vertical grouping is provided. The next most important display parameter shall be positioned under the primary display progressing down the panel with the least important at the bottom.

(a) A typical example of a military standard

5.1 *Subdivision of the display area*
In consideration of a simple, fast and accurate visual acquisition, the display area shall be divided into different sub-areas.
Such a division should be:

- Input area
- Output area
- Area for operational indications (such as status and alarms)

(b) From German standard DIN 66 234 Part 3 (1984), adapted from Smith [324]

5.15.3.2.1 *Standardization*
The content of displays within a system shall be presented in a consistent manner.

(c) From US military standard MIL-STD-1472C, revised (1983), adapted from Smith [324]

Figure 7.1 Sample design standards for displays. Adapted from Smith [324]. Copyright © 1986 IEEE

One component of the ISO standard 9241, pertaining to usability specification, applies equally to both hardware and software design. In the beginning of that document, the following definition of usability is given:

Usability The effectiveness, efficiency and satisfaction with which specified users achieve specified goals in particular environments.

Effectiveness The accuracy and completeness with which specified users can achieve specified goals in particular environments.

Efficiency The resources expended in relation to the accuracy and completeness of goals achieved.

Satisfaction The comfort and acceptability of the work system to its users and other people affected by its use.

The importance of such a definition in the standard is as a means of describing explicit measurements for usability. Such metrics can support usability engineering, as we saw in Chapter 6.

The strength of a standard lies in its ability to force large communities to abide – the so-called authority we have referred to earlier. It should be noted that such authority does not necessarily follow from the publication of a standard by a national or international body. In fact, many standards applying to software design are put forth as suggestive measures, rather than obligatory. The authority of a standard (or a guideline, for that matter) can only be determined from its use in practice. Some software products become de facto standards long before any formal standards document is published (for example, the X windowing system).

There is a much longer history of standards in safety-critical domains, such as nuclear power plants or aircraft design, where the consequences of poor design outweigh the expense of principled design. It is only as the perceived costs of unusable software in less safety-critical domains have become less acceptable that there has been a greater effort in developing standards for promoting usability.

7.4 GUIDELINES

We have observed that the incompleteness of theories underlying the design of interactive software makes it difficult to produce authoritative and specific standards. As a result, the majority of design rules for interactive systems are suggestive and more general guidelines. Our concern in examining the wealth of available guidelines is in determining their applicability to the various stages of design. The more abstract the guideline, the more it resembles the principles that we outlined in Section 7.2, which would be most suited to requirements specification. The more specific the guideline, the more suited it is to detailed design. The guidelines can also be automated to some extent, providing a direct means for translating detailed design specifications into actual implementation. There are a vast amount of published guidelines for interactive system design (they are frequently referred to as guidelines for user interface design). We will present only a few examples here to demonstrate the content of guidelines in that vast literature.

Several books and technical reports contain huge catalogs of guidelines. A classic example was a very general list compiled by Smith and Mosier in 1986 at the Mitre Corporation and sponsored by the Electronic Systems Division of the US Air Force [325]. The basic categories of the Smith and Mosier guidelines are:

1. Data Entry
2. Data Display
3. Sequence Control
4. User Guidance
5. Data Transmission
6. Data Protection

Each of these categories is further broken down into more specific subcategories which contain the particular guidelines. Figure 7.2 provides an example of the information contained in the Smith and Mosier guidelines. A striking feature of this compendium of guidelines is the extensive cross-referencing within the catalog, and citation to published work that supports each guideline. The Mitre Corporation has taken advantage of this structure and implemented the Smith and Mosier guidelines on a hypertext system, which provides rapid traversal of the network of guidelines to investigate the cross-references and citations.

1. **Data Entry**

1.1 *Position Designation*

1.1–1 **Distinctive Cursor**
For position designation on an electronic display, provide a movable cursor with distinctive visual features (shape, blink, etc.).

Exception When position designation involves only selection among displayed alternatives, highlighting selected items might be used instead of a separately displayed cursor.

Comment When choosing a cursor shape, consider the general content of the display. For instance, an underscore cursor would be difficult to see on a display of underscored text, or on a graphical display containing many other lines.

Comment If the cursor is changed to denote different functions (e.g. to signal deletion rather than entry), then each different cursor should be distinguishable from the others.

Comment If multiple cursors are used on the same display (e.g. one for alphanumeric entry and one for line drawing), then each cursor should be distinguishable from the others.

Reference Whitfield, Ball and Bird, 1983

See also	1.1–17	Distinctive multiple cursors
	4.0–9	Distinctive cursor

Figure 7.2 Sample guideline from Smith and Mosier [325], courtesy of The MITRE Corporation

Table 7.4 Comparison of dialog styles mentioned in guidelines

Smith and Mosier [325]	Mayhew [230]
Question and answer	Question and answer
Form filling	Fill-in forms
Menu selection	Menus
Function keys	Function keys
Command language	Command language
Query language	–
Natural language	Natural language
Graphic selection	Direct manipulation

A more recent, equally comprehensive catalog of general guidelines has been compiled by Mayhew [230]. Though this catalog is only in book form, and so limits the possibility of quick cross-referencing, this is one of the best sources for the experimental results which back the specific guidelines.

A major concern for all of the general guidelines is the subject of *dialog styles*, which in the context of these guidelines pertains to the means by which the user communicates input to the system, including how the system presents the communication device. Smith and Mosier identify eight different dialog styles and Mayhew identifies seven (see Table 7.4 for a comparison). The only real difference is the absence of query languages in Mayhew's list, but we can consider a query language as a special case of a command language. These interface styles have been described in more detail in Chapter 3.

Most guidelines are applicable for the implementation of any one of these dialog styles in isolation. It is also important to consider the possibility of mixing dialog styles in one application. In contrasting the action and language paradigms in Chapter 4, we concluded that it is not always the case that one paradigm wins over the other for all tasks in an application and, therefore, an application may want to mix the two paradigms. This equates to a mixing of dialog styles – a direct manipulation dialog being suitable for the action paradigm and a command language being suitable for the language paradigm. Mayhew provides guidelines and a technique for deciding how to mix dialog styles.

In moving from abstract guidelines to more specific and automated ones, it is necessary to introduce assumptions about the computer platform on which the interactive system is designed. So, for example, in Apple's *Human Interface Guidelines: the Apple Desktop Interface*, there is a clear distinction between the abstract guidelines (or principles), independent of the specific Macintosh hardware and software, and the concrete guidelines, which assume them. The abstract guidelines provide the so-called philosophy of programming that Apple would like designers to adopt in programming applications for the Macintosh. The more concrete guidelines are then seen as more concrete manifestations of that philosophy.

As an example, one abstract principle espoused in the Apple guidelines is *consistency*:

Effective applications are both consistent within themselves and consistent with one another.

We discussed consistency in Section 7.2 under the larger usability category of learnability, and the meaning in this context is similar. A more concrete directive that Apple provides is the ‘noun–verb’ ordering guideline: the user first selects an object (the noun) from the visible set on the Desktop and then selects an operation (the verb) to be applied to the object. For the sake of consistency, this ordering guideline is to be followed for all operation invocation involving the explicit and separate indication of an operation and the object or arguments of that operation.

Another less straightforward example from the Apple guidelines refers to user control:

The user, not the computer, initiates and controls all actions.

We considered issues of dialog initiative in Section 7.2 under the general usability category of flexibility. As we mentioned there, the issue of dialog initiative involves a trade-off between user freedom and system protection. In general, single-user computer systems operate in strict abidance of this guideline for user control; the user is allowed to initiate any dialog at all with the computer, whether or not it will have the intended result. Part of the success of direct manipulation interfaces lies in their ability to constrain user interaction to actions which are both syntactically correct (for example, preventing errors due to slips in typing) and will probably correspond to the intended user tasks.

Other popular graphical user interface (GUI) systems have published guidelines that describe how to adhere to abstract principles for usability in the narrower context of a specific programming environment. These guidelines are often referred to as *style guides* to reflect that they are not hard and fast rules, but suggested conventions for programming in that environment. Some examples are the OpenLook and the Open Software Foundation (OSF) Motif graphical user interfaces, both of which have published style guides [337, 275]. Programming in the style of these GUIs involves the use of toolkits which provide high-level widgets, as we have mentioned earlier in this book and will discuss in more detail in Chapter 8. More importantly, each of these GUIs has its own *look and feel*, which describes their expected behavior. The style guides are intended to help a programmer capture the elements of the look and feel of a GUI in her own programming. Therefore, style guides for the look and feel of a GUI promote the consistency within and between applications on the same computer platform.

We discussed menus in Chapter 3 as one of the major elements of the WIMP interface. As one example of a guideline for the design of menus, the OpenLook style guide suggests the following for grouping items in the same menu:

Use white space between long groups of controls on menus or in short groups when screen real estate is not an issue.

The justification for such a guideline is that the more options (or controls, as the term is used in the quoted guideline) on a menu, the longer it will take a user to locate and point to a desired item. As we discussed in Chapter 1, humans chunk related information in the learning process and this can be used to increase the efficiency of searching. Grouping of related items in a menu can supplement this chunking procedure. But be warned! Remember the scenario described in the Introduction to this book, in which we fell victim to closely grouped menu items which had drastically different effects in our word processor. Saving and deleting files might be considered logically similar since they both deal with operations on the file level. But simple slips made in pointing (which are all too easy with trackball devices) can change an intended save operation into an unintended and dangerous delete.

Worked exercise *Look up and report back guidelines for the use of color. Be able to state the empirical psychological evidence that supports the guidelines. Do the guidelines conflict with any other known guidelines? Which principles of interaction do they support?*

Answer There are many examples of guidelines for the use of color in the literature. Here are three good sources:

- C. Marlin Brown, *Human–Computer Interface Design Guidelines*, Ablex, 1988.
- Deborah J. Mayhew, *Principles and Guidelines in Software User Interface Design*, Prentice Hall, 1992.
- Sun Microsystems, Inc., *OpenLook Graphical User Interface Application Style Guidelines*, Addison-Wesley, 1990.

Taking an example from Mayhew, we have the following design guideline for the use of color as an informational cue for the user (for example, to inform the user that a string of text is a warning or error message):

> Do not use color without some other redundant cue.

Mayhew provides three reasons which empirically support this guideline:

1. Color may not be available on all machines on which the system is to be implemented. Therefore, if use of color is the only means to convey some important information to the user, then that information will be lost in a monochrome (no color) system. Redundant color coding will allow for portability across different computing platforms.
2. Empirical evidence shows that 8% of the (general) male population and 0.4% of the female population has some color deficiency, so they cannot accurately recognize or distinguish between various colors. Again, if color is the only means for conveying some information, this significant portion of the user population will be slighted.
3. It has been shown that redundant color coding enhances user performance

This guideline supports several of the principles discussed in this chapter:

Substitutivity The system is able to substitute color-coded information and other means (for example, text, sound) to represent some important information. We could turn the argument around and suggest that the user be able to provide color input (by selecting from a palette menu) or other forms of input to provide relevant information to the system.

Observability This principle is all about the system being able to provide the user with enough information about its internal state to assist his task. Relying strictly on color-coded information, as pointed out above, could reduce the observability of a system for some users.

Synthesis If a change in color is used to indicate the changing status of some system entity (perhaps a change in temperature above a threshold value is signalled by an icon becoming red), those who cannot detect the change in color would be deprived of this information. Synthesis is about supporting the user's ability to detect such significant changes, especially when they are a result of previous user actions.

There is no evidence of existing guidelines that this particular guideline for color violates.

Another example of a color guideline (found in all three of the above references) is the demand to consider cultural information in the selection of particular colors. For example, Mayhew states that western cultures tend to interpret green to mean go or safe; red to mean stop, on, hot or emergency; and blue to mean cold or off. Using color to suggest these kinds of meanings is in support of the familiarity principle within learnability. However, in other cultures different meanings may be associated with these colors, as we saw in Chapter 3, and consistent use of color (another guideline) might lead to confusion. Hence, strict adherence to this guideline would suggest a violation of the consistency of color application guideline. However, if consistency is applied relative to the meaning of the color (as opposed to its actual color), this guideline would not have to conflict.

7.5 GOLDEN RULES AND HEURISTICS

So far we have considered a range of abstract principles and detailed guidelines, which can be used to help designers produce more usable systems. But all of these rules require a certain amount of commitment on the part of the designer, either to track down appropriate guidelines or to interpret principles. Is there a simpler way?

A number of advocates of user-centered design have presented sets of 'golden rules' or heuristics. While these are inevitably 'broad-brush' design rules, which may not be always be applicable to every situation, they do provide a useful checklist or summary of the essence of design advice. It is clear that any designer following even these simple rules will produce a better system than one who ignores them.

There are many sets of heuristics, but the most well used are Nielsen's ten heuristics, Shneiderman's eight golden rules and Norman's seven principles. Nielsen's heuristics are intended to be used in evaluation and will therefore be discussed in Chapter 9. We will consider the other two sets here.

7.5.1 Shneiderman's Eight Golden Rules of Interface Design

Shneiderman's eight golden rules provide a convenient and succinct summary of the key principles of interface design. They are intended to be used during design but can also be applied, like Nielsen's heuristics, to the evaluation of systems. Notice how they relate to the abstract principles discussed earlier.

1. *Strive for consistency* in action sequences, layout, terminology, command use and so on.
2. *Enable frequent users to use shortcuts*, such as abbreviations, special key sequences and macros, to perform regular, familiar actions more quickly.
3. *Offer informative feedback* for every user action, at a level appropriate to the magnitude of the action.
4. *Design dialogs to yield closure* so that the user knows when they have completed a task.
5. *Offer error prevention and simple error handling* so that, ideally, users are prevented from making mistakes and, if they do, they are offered clear and informative instructions to enable them to recover.
6. *Permit easy reversal of actions* in order to relieve anxiety and encourage exploration, since the user knows that he can always return to the previous state.
7. *Support internal locus of control* so that the user is in control of the system, which responds to his actions.
8. *Reduce short-term memory load* by keeping displays simple, consolidating multiple page displays and providing time for learning action sequences.

These rules provide a useful shorthand for the more detailed sets of principles described earlier. Like those principles, they are not applicable to every eventuality and need to be interpreted for each new situation. However, they are broadly useful and their application will only help most design projects.

7.5.2 Norman's Seven Principles for Transforming Difficult Tasks into Simple Ones

In Chapter 3 we discussed Norman's execution–evaluation cycle, in which he elaborates the seven stages of action. Later, in his classic book *The Design of Everyday Things*, he summarizes user-centered design using the following seven principles:

1. *Use both knowledge in the world and knowledge in the head*. People work better when the knowledge they need to do a task is available externally – either explicitly or through the constraints imposed by the environment. But experts also need to be able to internalize regular tasks to increase their efficiency. So systems should provide the necessary knowledge within the environment and their operation should be transparent to support the user in building an appropriate mental model of what is going on.
2. *Simplify the structure of tasks*. Tasks need to be simple in order to avoid complex problem solving and excessive memory load. There are a number of ways to simplify the structure of tasks. One is to provide mental aids to help the user keep track of stages in a more complex task. Another is to use technology to provide the user with more information about the task and better feedback. A third approach is to automate the task or part of it, as long as this does not detract from the user's experience. The final approach to simplification is to change the nature

of the task so that it becomes something more simple. In all of this, it is important not to take control away from the user.

3. *Make things visible*: bridge the gulfs of execution and evaluation. The interface should make clear what the system can do and how this is achieved, and should enable the user to see clearly the effect of their actions on the system.
4. *Get the mappings right*. User intentions should map clearly onto system controls. User actions should map clearly onto system events. So it should be clear what does what and by how much. Controls, sliders and dials should reflect the task – so a small movement has a small effect and a large movement a large effect.
5. *Exploit the power of constraints*, both natural and artificial. Constraints are things in the world that make it impossible to do anything but the correct action in the correct way. A simple example is a jigsaw puzzle, where the pieces only fit together in one way. Here the physical constraints of the design guide the user to complete the task.
6. *Design for error*. To err is human, so anticipate the errors the user could make and design recovery into the system.
7. *When all else fails, standardize*. If there are no natural mappings then arbitrary mappings should be standardized so that users only have to learn them once. It is this standardization principle that enables drivers to get into a new car and drive it with very little difficulty – key controls are standardized. Occasionally one might switch on the indicator lights instead of the windscreen wipers, but the critical controls (accelerator, brake, clutch, steering) are always the same.

Norman's seven principles provide a useful summary of his user-centered design philosophy but the reader is encouraged to read the complete text of *The Design of Everyday Things* to gain the full picture.

7.6 HCI PATTERNS

As we observed in Chapter 4, one way to approach design is to learn from examples that have proven to be successful in the past: to reuse the knowledge of what made a system – or paradigm – successful. Patterns are an approach to capturing and reusing this knowledge – of abstracting the essential details of successful design so that these can be applied again and again in new situations.

Patterns originated in architecture, where they have been used successfully, and they are also used widely in software development to capture solutions to common programming problems. More recently they have been used in interface and web design.

A pattern is an invariant solution to a recurrent problem within a specific context. Patterns address the problems that designers face by providing a 'solution statement'. This is best illustrated by example. Alexander, who initiated the pattern concept, proposes a pattern for house building called 'Light on Two Sides of Every Room'. The problem being addressed here is that

> When they have a choice, people will always gravitate to those rooms which have light on two sides, and leave the rooms which are lit only from one side unused and empty.

The proposed solution is to provide natural light from two sides of every room:

> Locate each room so that it has outdoor space outside it on at least two sides, and then place windows in these outdoor walls so that natural light falls into every room from more than one direction [9a, pattern 159].

Note that the solution says nothing about where these windows should be located or at what angle they should be to each other. A room with windows on opposite walls, or at right angles, or with a window and a skylight would all fulfill the pattern. Patterns capture only the invariant properties of good design – the common elements that hold between all instances of the solution. The specific implementation of the pattern will depend on the circumstance and the designer's creativity.

There are many examples of HCI patterns, and the interested reader is referred to pattern collections and languages such as [345, 37, 356] and the Pattern Gallery, which illustrates some of the various forms used in HCI patterns [132]. A well-known example, 'go back to a safe place', adapted from Tidwell's Common Ground collection, is given as an illustration (Figure 7.3). This is quite a low-level interface pattern, but patterns can also address high-level issues such as organizational structures or cooperative groups. As you can see, the pattern states the problem and the solution but also includes a rationale, explaining where the pattern has come from and in what context it applies, and examples to illustrate the pattern.

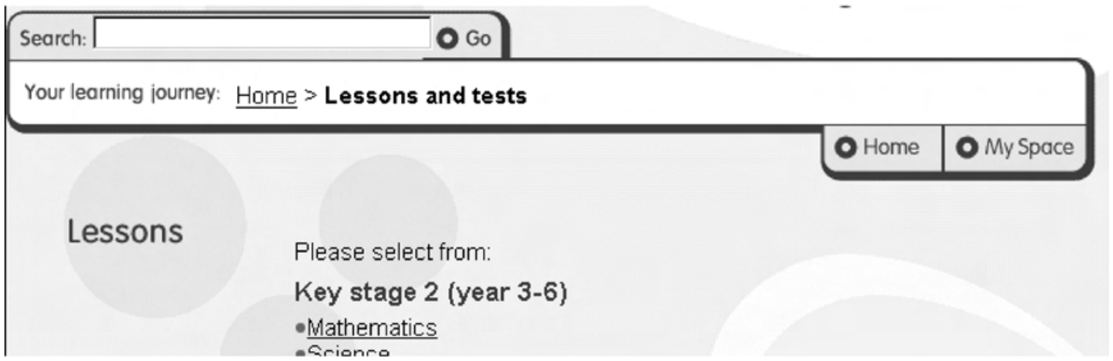
The pattern also has references to other patterns, indicating both the context in which it can be applied (the top references) and the patterns that may be needed to complete it (the bottom references). This connects the patterns together into a *language*. Patterns in isolation have limited use, but by traversing the hierarchy, through these references, the user is assisted in generating a complete design.

Patterns and pattern languages are characterized by a number of features, which, taken as a whole, distinguish them from other design rules:

- They capture design practice and embody knowledge about successful solutions: they come from practice rather than psychological theory.
- They capture the essential common properties of good design: they do not tell the designer *how* to do something but what needs to be done and why.
- They represent design knowledge at varying levels, ranging from social and organizational issues through conceptual design to detailed widget design.
- They are not neutral but embody values within their rationale. Alexander's language clearly expresses his values about architecture. HCI patterns can express values about what is humane in interface design.
- The concept of a pattern language is generative and can therefore assist in the development of complete designs.
- They are generally intuitive and readable and can therefore be used for communication between all stakeholders.

Patterns are a relatively recent addition to HCI representations, in which there are still many research issues to resolve. For instance, it is not clear how patterns can best be identified or how languages should be structured to reflect the temporal concerns of interaction. However, the recent publication of a complete pattern language for web design [356], aimed at commercial designers, may mark a turning point and see a more widespread adoption of the approach in interface design.

... NAVIGABLE SPACES or STEP BY STEP and a CONTROL PANEL are in place, which require the user to be able to move through the steps page by page.



From (www.learn.co.uk). An example of a toolbar that provides the option to return home and keeps track of the learning journey, providing the option to link back to a safe place.

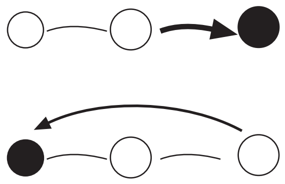
❖ ❖ ❖

It is easy to get lost in the tangle of links in a website.

People don't use the web like a TV or magazine. They use the web to find what they are looking for and then stop. They may select a wrong link and not be able to find their way back to something relevant. They do not always keep track of where they've been. They may forget where they have been if they are interrupted when using the site.

You are more likely to explore a website if you are sure that you can easily get out of an undesired state or space; that assurance engenders a feeling of security. Backtracking out of a long navigation path can be very tedious.

Therefore:
Always include a way back to a place that acts as a 'vantage point' from where you can reorientate yourself.



This pattern can be used in conjunction with GO BACK ONE STEP and CONTINUE TO NEXT STEP.

Figure 7.3 An example pattern 'go back to a safe place' adapted from Tidwell's Common Ground collection. Courtesy of Jenifer Tidwell

7.7 SUMMARY

We have seen how design rules can be used to provide direction for the design process, although the more general and interesting the design rule is for promoting usability, the further away it is from the actual language of design.

We have considered abstract principles, standards and guidelines, golden rules and heuristics, and patterns, and have looked at examples of each. The most abstract design rules are principles, which represent generic knowledge about good design practice. Standards and guidelines are more specific. Standards have the highest authority, being set by national or international bodies to ensure compliance by a large community. Guidelines are less authoritative but offer specific contextual advice, which can inform detailed design. Heuristics and 'golden rules' are succinct collections of design principles and advice that are easily assimilated by any designer. Patterns capture design practice and attempt to provide a generative structure to support the design process.

EXERCISES

7.1 What was the problem with the synthesis example comparing a command language interface with a visual interface? Can you suggest a fix to make a visual interface really immediately honest?

7.2 It has been suggested in this chapter that consistency could be considered a major category of interactive principles, on the same level as learnability, flexibility and robustness. If this was the case, which principles discussed in this chapter would appear in support of consistency?

7.3 Find as much information as you can on ISO standards that relate to usability. (**Hint:** Many standards are discussed in terms of ergonomics.) How many different standards and draft standards can you find?

7.4 Can you think of any instances in which the 'noun–verb' guideline for operations, as suggested in the Apple human interface guidelines for the Desktop Interface, would be violated? Suggest other abstract guidelines or principles besides consistency which support your example. (**Hint:** Think about moving files around on the Desktop.)

7.5 Can you think of any instances in which the user control guideline suggested by Apple is not followed? (**Hint:** Think about the use of dialog boxes.)

7.6 Find a book on guidelines. List the guidelines that are provided and classify them in terms of the activity in the software life cycle to which they would most likely apply.

7.7 (a) Distinguish between principles, guidelines and standards, using examples of each to illustrate.
(b) Why is context important in selecting and applying guidelines and principles for interface design? Illustrate your answer with examples.

7.8 (a) Why are there few effective HCI standards?
(b) How do 'golden rules' and heuristics help interface designers take account of cognitive psychology? Illustrate your answer with examples.

7.9 Using the web design pattern language in *The Design of Sites* [356] produce a design for an e-commerce site for a small retail business. How well does the language support the design process?

RECOMMENDED READING

H. Thimbleby, Design of interactive systems. In J. A. McDermid, editor, *The Software Engineer's Reference Book*, Chapter 57, Butterworth–Heinemann, 1992.
Thimbleby provides a very insightful list of general principles which apply to interactive systems. Some of the principles we have described in this chapter come from Thimbleby's work, though we have concentrated more on providing an overall organizational framework for the principles.

T. Stewart, Ergonomics user interface standards: are they more trouble than they are worth?, *Ergonomics*, Vol. 43, No. 7, pp. 1030–44, July 2000.
A review of the development of user interface standards, which considers the history of the endeavour, its successes and failures.

D. J. Mayhew, *Principles and Guidelines in Software User Interface Design*, Prentice Hall, 1992.
A comprehensive catalog of general interactive system guidelines which provides much of the experimental evidence to support specific guidelines.

B. Shneiderman, *Designing the User Interface*, 3rd edition, Addison-Wesley, 1998.
The source of Eight Golden Rules.

D. Norman, *The Design of Everyday Things*, MIT Press, 1998.
Classic exposition of design, including the seven principles.

D. Van Duyne, J. Landay and J. Hong, *The Design of Sites: Patterns, Principles and Processes for Crafting a Customer-centred Web Experience*, Addison-Wesley, 2003.
The first substantial and widely applicable HCI complete pattern language (for web design) presented accessibly for use in design teams.

The HCI Service, *HCI Tools & Methods Handbook*, 1991.
This booklet was produced under the United Kingdom Department of Trade and Industry (DTI) *Usability Now!* program. It contains a list of books on guidelines as well as a summary of the available user-centered design techniques as they can be applied to the software life cycle. The booklet and other information about the program is available from the HCI Service, PO Box 31, Loughborough, Leicestershire LE11 1QU, United Kingdom.

8 Implementation support

OVERVIEW

- Programming tools for interactive systems provide a means of effectively translating abstract designs and usability principles into an executable form. These tools provide different levels of services for the programmer.
- Windowing systems are a central environment for both the programmer and user of an interactive system, allowing a single workstation to support separate user–system threads of action simultaneously.
- Interaction toolkits abstract away from the physical separation of input and output devices, allowing the programmer to describe behaviors of objects at a level similar to how the user perceives them.
- User interface management systems are the final level of programming support tools, allowing the designer and programmer to control the relationship between the presentation objects of a toolkit with their functional semantics in the actual application.

8.1 INTRODUCTION

In this chapter, we will discuss the programming support that is provided for the implementation of an interactive system. We have spent much effort up to this point considering design and analysis of interactive systems from a relatively abstract perspective. We did this because it was not necessary to consider the specific details of the devices used in the interaction. Furthermore, consideration of that detail was an obstacle to understanding the interaction from the user's perspective. But we cannot forever ignore the specifics of the device. It is now time to devote some attention to understanding just how the task of coding the interactive application is structured.

The detailed specification gives the programmer instructions as to what the interactive application must do and the programmer must translate that into machine executable instructions to say how that will be achieved on the available hardware devices. The objective of the programmer then is to translate down to the level of the software that runs the hardware devices. At its crudest level, this software provides the ability to do things like read events from various input devices and write primitive graphics commands to a display. Whereas it is possible in that crude language to produce highly interactive systems, the job is very tedious and highly error prone, amenable to computer hackers who relish the intricacy and challenge but not necessarily those whose main concern is the design of very usable interactive systems.

The programming support tools which we describe in this chapter aim to move that executable language up from the crudely expressive level to a higher level in which the programmer can code more directly in terms of the interaction objects of the application. The emphasis here is on how building levels of abstraction on top of the essential hardware and software services allows the programmer to build the system in terms of its desired *interaction techniques*, a term we use to indicate the intimate relationship between input and output. Though there is a fundamental separation between input and output devices in the hardware devices and at the lowest software level, the distinction can be removed at the programming level with the right abstractions and hiding of detail.

In the remainder of this chapter, we will address the various layers which constitute the move from the low-level hardware up to the more abstract programming concepts for interaction. We begin in Section 8.2 with the elements of a windowing system, which provide for device independence and resource sharing at the programming level. Programming in a window system frees the programmer from some of the worry about the input and output primitives of the machines the application will run on, and allows her to program the application under the assumption that it will receive a stream of event requests from the window manager. In Section 8.3 we describe the two fundamental ways this stream of events can be processed to link the interface with the application functionality: by means of a read–evaluation control loop internal to the application program or by a centralized notification-based technique external to it. In Section 8.4, we describe the use of toolkits as mechanisms to link input and output at the programming level. In Section 8.5, we discuss the large class of development tools lumped under the categories of user interface management systems, or UIMS, and user interface development systems, UIDS.

8.2 ELEMENTS OF WINDOWING SYSTEMS

In earlier chapters, we have discussed the elements of the WIMP interface but only with respect to how they enhance the interaction with the end-user. Here we will describe more details of windowing systems used to build the WIMP interface.

The first important feature of a windowing system is its ability to provide programmer independence from the specifics of the hardware devices. A typical workstation will involve some visual display screen, a keyboard and some pointing device, such as a mouse. Any variety of these hardware devices can be used in any interactive system and they are all different in terms of the data they communicate and the commands that are used to instruct them. It is imperative to be able to program an application that will run on a wide range of devices. To do this, the programmer wants to direct commands to an *abstract terminal*, which understands a more generic language and can be translated to the language of many other specific devices. Besides making the programming task easier, the abstract terminal makes portability of application programs possible. Only one translation program – or *device driver* – needs to be written for a particular hardware device and then any application program can access it.

A given windowing system will have a fixed generic language for the abstract terminal which is called its *imaging model*. The imaging models are sufficient to describe very arbitrary images. For efficiency reasons, specific primitives are used to handle text images, either as specific pixel images or as more generic font definitions.

Examples of imaging models

Pixels
The display screen is represented as a series of columns and rows of points – or pixels – which can be explicitly turned on or off, or given a color. This is a common imaging model for personal computers and is also used by the X windowing system.

Graphical kernel system (GKS)
An international standard which models the screen as a collection of connected segments, each of which is a macro of elementary graphics commands.

Programmer's hierarchical interface to graphics (PHIGS)
Another international standard, based on GKS but with an extension to model the screen as editable segments.

PostScript
A programming language developed by Adobe Corporation which models the screen as a collection of paths which serve as infinitely thin boundaries or stencils which can be filled in with various colors or textured patterns and images.

Though these imaging models were initially defined to provide abstract languages for output only, they can serve at least a limited role for input as well. So, for example, the pixel model can be used to interpret input from a mouse in terms of the pixel coordinate system. It would then be the job of the application to process the input event further once it knows where in the image it occurred. The other models above can provide even more expressiveness for the input language, because they can relate the input events to structures that are identifiable by the application program. Both PHIGS and PostScript have been augmented to include a more explicit model of input.

When we discussed the WIMP interface as an interaction paradigm in Chapter 4, we pointed out its ability to support several separate user tasks simultaneously. Windowing systems provide this capability by sharing the resources of a single hardware configuration with several copies of an abstract terminal. Each abstract terminal will behave as an independent *process* and the windowing system will coordinate the control of the concurrent processes. To ease the programming task again, this coordination of simultaneously active processes can be factored out of the individual applications, so that they can be programmed as if they were to operate in isolation. The window system must also provide a means of displaying the separate applications, and this is accomplished by dedicating a region of the display screen to each active abstract terminal. The coordination task then involves resolving display conflicts when the visible screen regions of two abstract terminals overlap.

In summary, we can see the role of a windowing system, depicted in Figure 8.1, as providing

independence from the specifics of programming separate hardware devices;

management of multiple, independent but simultaneously active applications.

Next, we discuss the possible architectures of a windowing system to achieve these two tasks.

8.2.1 Architectures of windowing systems

Bass and Coutaz [29] identify three possible architectures for the software to implement the roles of a windowing system. All of them assume that device drivers are separate from the application programs. The first option is to implement and replicate the management of the multiple processes within each of the separate applications. This is not a very satisfactory architecture because it forces each application to consider the difficult problems of resolving synchronization conflicts with the shared hardware devices. It also reduces the portability of the separate applications. The second option is to implement the management role within the kernel of the operating system, centralizing the management task by freeing it from the individual applications. Applications must still be developed with the specifics of the particular operating system in mind. The third option provides the most portability, as the management function is written as a separate application in its own right and

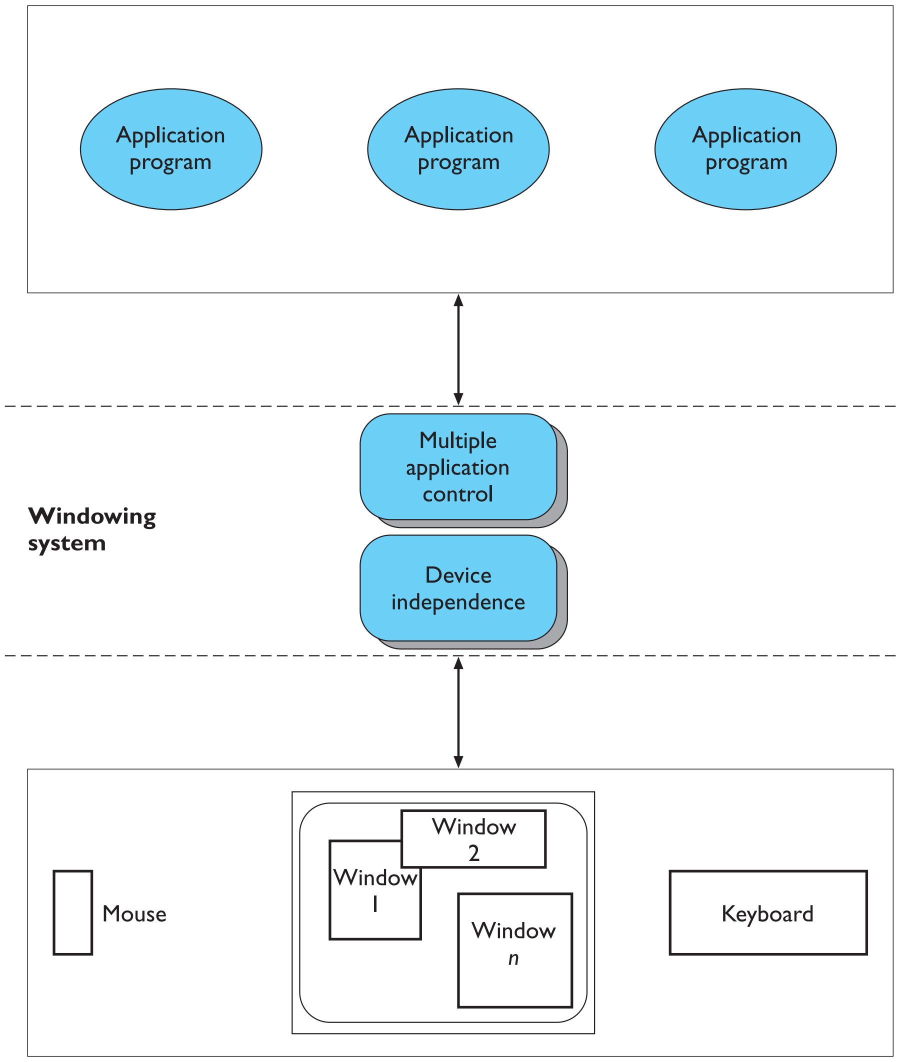


Figure 8.1 The roles of a windowing system

so can provide an interface to other application programs that is generic across all operating systems. This final option is referred to as the *client–server architecture*, and is depicted in Figure 8.2.

In practice, the divide among these proposed architectures is not so clear and any actual interactive application or set of applications operating within a window system may share features with any one of these three conceptual architectures. Therefore, it may have one component that is a separate application or process together with some built-in operating system support and hand-tuned application

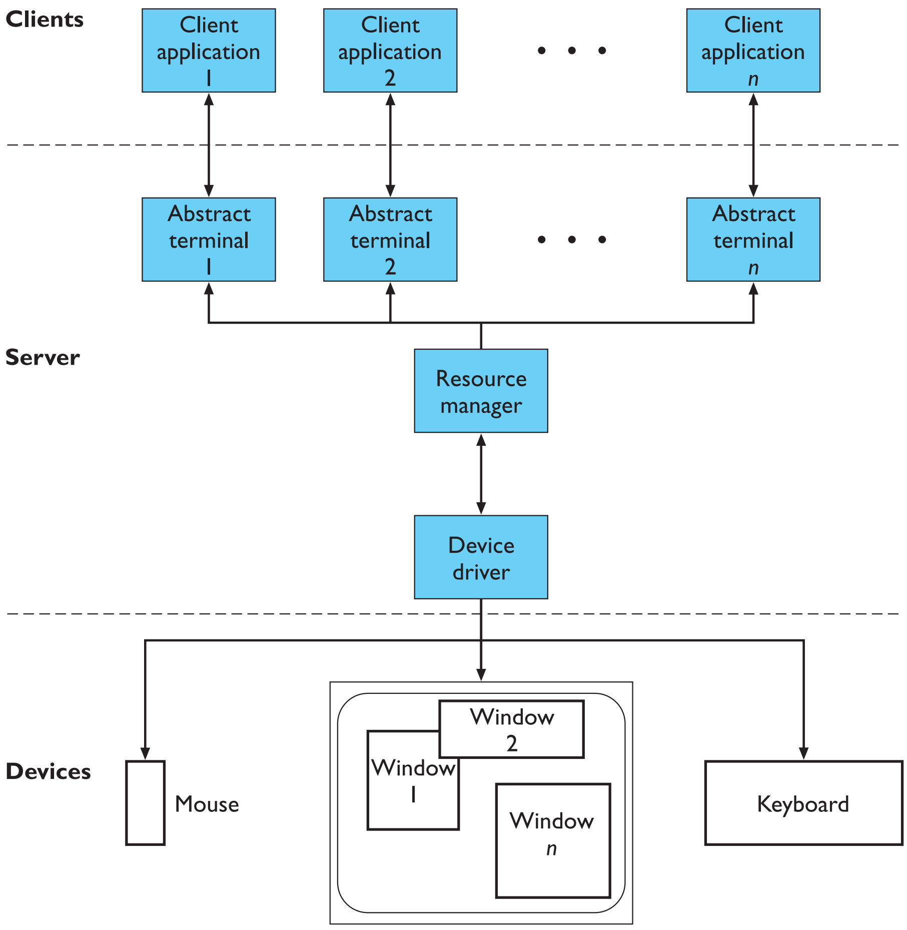


Figure 8.2 The client–server architecture

support to manage the shared resources. So applications built for a window system which is notionally based on the client–server model may not be as portable as one would think.

A classic example of a window system based on the client–server architecture is the industry-standard X Window System (Release 11), developed at the Massachusetts Institute of Technology (MIT) in the mid-1980s. Figure 8.3 shows the software architecture of X. X (or X11), as we mentioned earlier, is based on a pixel-based imaging model and assumes that there is some pointing mechanism available. What distinguishes X from other window systems, and the reason it has been adopted as a standard, is that X is based on a network protocol which clearly defines the server–client communication. The *X Protocol* can be implemented on different computers and operating systems, making X more device independent. It also means that client and server need not even be on the same system in order to communicate to

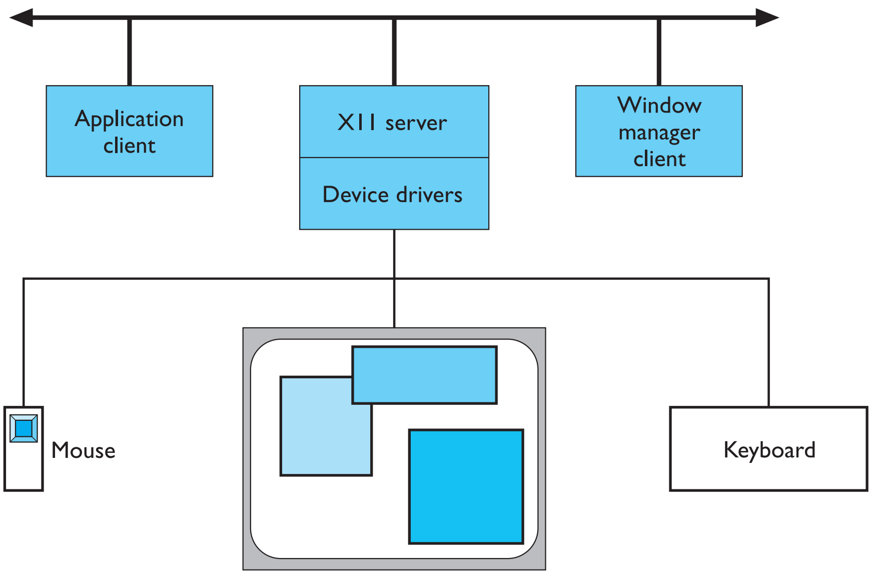


Figure 8.3 The X Window System (Release 11) architecture

the server. Each client of the X11 server is associated to an abstract terminal or main window. The X server performs the following tasks:

- allows (or denies) access to the display from multiple client applications;
- interprets requests from clients to perform screen operations or provide other information;
- demultiplexes the stream of physical input events from the user and passes them to the appropriate client;
- minimizes the traffic along the network by relieving the clients from having to keep track of certain display information, like fonts, in complex data structures that the clients can access by ID numbers.

A separate client – the *window manager* – enforces policies to resolve conflicting input and output requests to and from the other clients. There are several different window managers which can be used in X, and they adopt different policies. For example, the window manager would decide how the user can change the focus of his input from one application to another. One option is for the user to nominate one window as the active one to which all subsequent input is directed. The other option is for the active window to be implicitly nominated by the position of the pointing device. Whenever the pointer is in the display space of a window, all input is directed to it. Once the pointer is moved to a position inside another window, that window becomes active and receives subsequent input. Another example of window manager policy is whether visible screen images of the client windows can overlap or must be non-overlapping (called tiling). As with many other windowing systems, the client applications can define their own hierarchy of subwindows, each of which is constrained to the coordinate space of the parent window. This subdivision of the main client window allows the programmer to manage the input and output for a single application similar to the window manager.

To aid in the design of specific window managers, the X Consortium has produced the *Inter-Client Communication Conventions Manual* (*ICCCM*), which provides conventions for various policy issues that are not included in the X definition. These policies include:

- rules for transferring data between clients;
- methods for selecting the active client for input focus;
- layout schemes for overlapping/tiled windows as screen regions.

8.3 PROGRAMMING THE APPLICATION

We now concentrate our attention on programming the actual interactive application, which would correspond to a client in the client–server architecture of Figure 8.2. Interactive applications are generally user driven in the sense that the action the application takes is determined by the input received from the user. We describe two programming paradigms which can be used to organize the flow of control within the application. The windowing system does not necessarily determine which of these two paradigms is to be followed.

The first programming paradigm is the *read–evaluation loop*, which is internal to the application program itself (see Figure 8.4). Programming on the Macintosh follows this paradigm. The server sends user inputs as structured events to the client application. As far as the server is concerned, the only importance of the event is the client to which it must be directed. The client application is programmed to read any event passed to it and determine all of the application-specific behavior that results as a response to it. The logical flow of the client application is indicated in the leftmost box of Figure 8.4. In pseudocode the read–evaluation loop would look like the following:

```
repeat
        read-event(myevent)
        case   myevent.type
               type_1 :
                         do type_1 processing
               type_2 :
                         do type_2 processing
               .
               .
               .
```

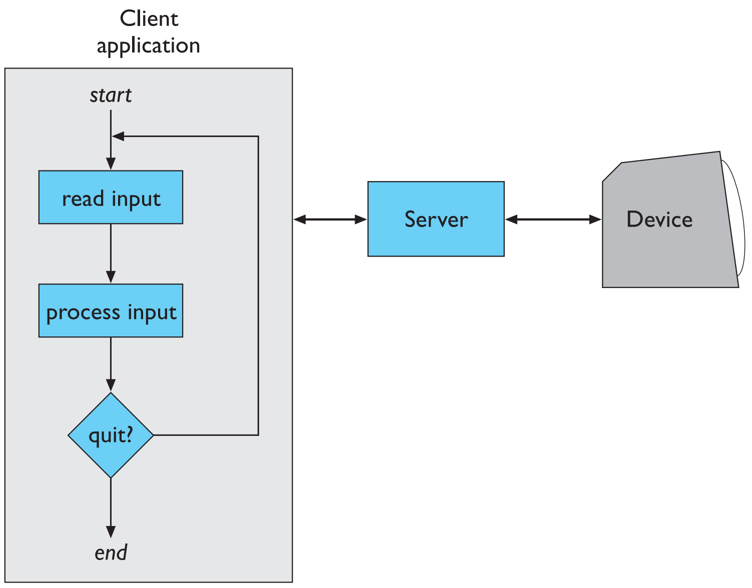


Figure 8.4 The read–evaluate loop paradigm

```
              type_n :
                   do type_n processing
         end case
    end repeat
```

The application has complete control over the processing of events that it receives. The downside is that the programmer must execute this control over every possible event that the client will receive, which could prove a very cumbersome task. On the Macintosh, this process can be aided somewhat by programming tools, such as MacApp, which automate some of the tedium.

The other programming paradigm is *notification based*, in which the main control loop for the event processing does not reside within the application. Instead, a centralized *notifier* receives events from the window system and filters them to the application program in a way declared by the program (see Figure 8.5). The application program informs the notifier what events are of interest to it, and for each event declares one of its own procedures as a *callback* before turning control over to the notifier. When the notifier receives an event from the window system, it sees if that event was identified by the application program and, if so, passes the event and control over to the callback procedure that was registered for the event. After processing, the callback procedure returns control to the notifier, either telling it to continue receiving events or requesting termination.

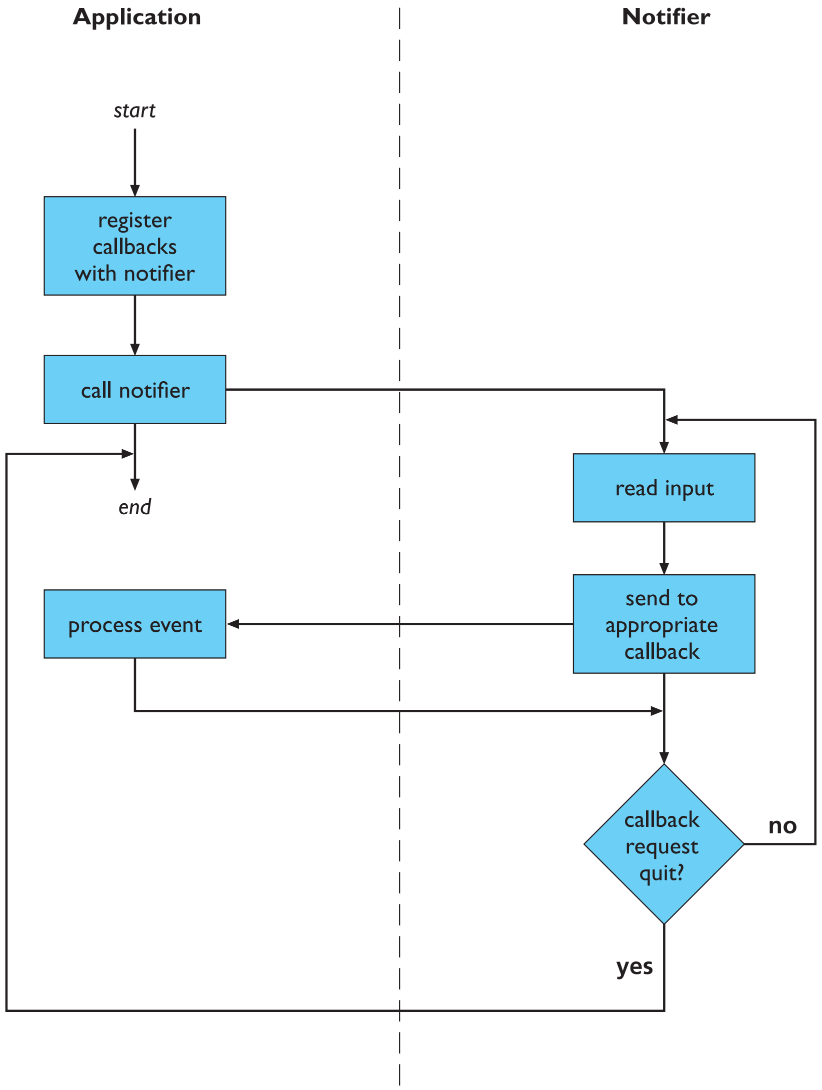


Figure 8.5 The notification-based programming paradigm

Control flow is centralized in the notifier, which relieves the application program of much of the tedium of processing every possible event passed to it by the window system. But this freedom from control does not come without a price. Suppose, for example, that the application program wanted to produce a pre-emptive dialog box, perhaps because it has detected an error and wants to obtain confirmation from the user before proceeding. The pre-emptive dialog effectively discards all subsequent user actions except for ones that it requires, say selection by the user inside a certain region of the screen. To do this in the read–evaluation paradigm is fairly

Example: a notification-based program

Figure 8.6 provides an example of notification-based programming in C using the XView toolkit (toolkits are described in the next section).

```
/*
 * quit.c -- simple program to display a panel button that says "Quit".
 * Selecting the panel button exits the program.
 */
# include <xview/xview.h>
# include <xview/frame.h>
# include <xview/panel.h>

Frame frame;

main  (argc, argv)
int argc;
char *argv[];
{
    Panel panel;
    void quit();

    xv_init(XV_INIT_ARGC_PTR_ARGV, &argc, argv, NULL);

    frame = (Frame) xv_create(NULL, FRAME,
        FRAME_LABEL,     argv[0],
        XV_WIDTH,        200,
        XV_HEIGHT,       100,
        NULL);

    panel = (Panel) xv_create(frame, PANEL, NULL);

    (void) xv_create(panel, PANEL_BUTTON,
             PANEL_LABEL_STRING,      "Quit",
             PANEL_NOTIFY_PROC,        quit,
             NULL);

    xv_main_loop(frame);
    exit(0);
}

void quit()
{
    xv_destroy_safe(frame);
}
```

Figure 8.6 A simple program to demonstrate notification-based programming. Example taken from Dan Heller [169], reproduced by permission of O'Reilly and Associates, Inc

The program produces a window, or frame, with one button, labeled `Quit`, which when selected by the pointer device causes the program to quit, destroying the window (see Figure 8.7 for the screen image produced by the sample program `quit.c`). Three objects are created in this program: the outermost frame, a panel within that frame and the button in the panel. The procedure `xv_create`, used on lines 17, 22 and 23 in the source code of Figure 8.6, is used by the application program to register the objects with the XView notifier. In the last instance on line 23, the application programmer informs the notifier of the callback procedure to be invoked when the object, a button, is selected. The application program then initiates the notifier by the procedure call `xv_main_loop`. When the notifier receives a select event for the button, control is passed to the procedure `quit` which destroys the outermost frame and requests termination.

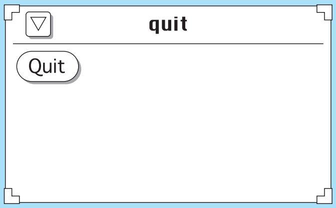


Figure 8.7 Screen image produced by sample program `quit.c`

straightforward. Suppose the error condition occurred during the processing of an event of type `type_2`. Once the error condition is recognized, the application then begins another read–evaluation loop contained within that branch of the **case** statement. Within that loop, all non-relevant events can be received and discarded. The pseudocode example given earlier would be modified in the following way:

```
repeat
      read-event(myevent)
      case   myevent.type
             type_1  :
                     do type_1 processing
             type_2  :
                     . . .
                     if      (error-condition) then
                             repeat
                                    read-event(myevent2)
                                    case   myevent2.type
```

```
                                          type_1  :
                                          .
                                          .
                                          .
                                          type_n  :
                                   end case
                            until (end-condition2)
                    end if
                    . . .
             .
             .
             .
             type_n :
                    do type_n processing
      end case
until (end-condition)
```

In the notification-based paradigm, such a pre-emptive dialog would not be so simple, because the control flow is out of the hands of the application programmer. The callback procedures would all have to be modified to recognize the situations in which the pre-emptive dialog is needed and in those situations disregard all events which are passed to them by the notifier. Things would be improved, however, if the application programmer could in such situations access the notifier directly to request that previously acceptable events be ignored until further notice.

Design Focus

Going with the grain

It is possible to use notification-based code to produce pre-emptive interface dialog such as a *modal dialog box*, but much more difficult than with an event-loop-based system. Similarly, it is possible to write event-loop-based code which is not pre-emptive, but again it is difficult to do so. If you are not careful, systems built using notification-based code will have lots of non-modal dialog boxes and vice versa. Each programming paradigm has a *grain*, a tendency to push you towards certain kinds of interface.

If you know that the interface you require fits more closely to one paradigm or another then it is worth selecting the programming paradigm to make your life easier! Often, however, you do not have a choice. In this case you have to be very careful to decide what kind of interface dialog you want *before* you (or someone else) start coding. Where the desired interface fits the grain of the paradigm you don't have to worry. Where the desired behavior runs against the grain you must be careful, both in coding and testing as these are the areas where things will go wrong.

Of course, if you don't *explicitly* decide what behavior you want or you specify it unclearly, then it is likely that the resulting system will simply run with the grain, whether or not that makes a good interface.

8.4 USING TOOLKITS

As we discussed in Chapter 4, a key feature of WIMP interfaces from the user's perspective is that input and output behaviors are intrinsically linked to independent entities on the display screen. This creates the illusion that the entities on the screen are the objects of interest – interaction objects we have called them – and that is necessary for the action world of a direct manipulation interface. A classic example is the mouse as a pointing device. The input coming from the hardware device is separate from the output of the mouse cursor on the display screen. However, since the visual movement of the screen cursor is linked with the physical movement of the mouse device, the user feels as if he is actually moving the visual cursor. Even though input and output are actually separate, the illusion causes the user to treat them as one; indeed, both the visual cursor and the physical device are referred to simply as 'the mouse'. In situations where this link is broken, it is easy to see the user's frustration.

In Figure 8.8, we show an example of how input and output are combined for interaction with a button object. As the user moves the mouse cursor over the button, it changes to a finger to suggest that the user can push it. Pressing the mouse button down causes the button to be highlighted and might even make an audible click like the keys on some keyboards, providing immediate feedback that the button has been pushed. Releasing the mouse button unhighlights the button and moving the mouse off the button changes the cursor to its initial shape, indicating that the user is no longer over the active area of the button.

From the programmer's perspective, even at the level of a windowing system, input and output are still quite separate for everything except the mouse, and it takes quite a bit of effort in the application program to create the illusion of the interaction object such as the button we have just described. To aid the programmer in fusing input and output behaviors, another level of abstraction is placed on top of the window system – the *toolkit*. A toolkit provides the programmer with a set of ready-made interaction objects – alternatively called interaction techniques, gadgets or widgets – which she can use to create her application programs. The interaction

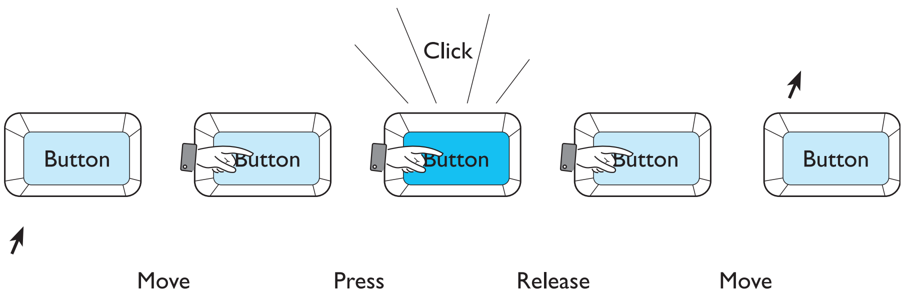


Figure 8.8 Example of behavior of a button interaction object

objects have a predefined behavior, such as that described for the button, that comes for free without any further programming effort. Toolkits exist for all windowing environments (for example, OSF/Motif and XView for the X Window system, the Macintosh Toolbox and the Software Development Toolkit for Microsoft Windows).

To provide flexibility, the interaction objects can be tailored to the specific situation in which they are invoked by the programmer. For example, the label on the button could be a parameter which the programmer can set when a particular button is created. More complex interaction objects can be built up from smaller, simpler ones. Ultimately, the entire application can be viewed as a collection of interaction objects whose combined behavior describes the semantics of the whole application.

The sample program `quit.c` in Figure 8.6 uses the XView toolkit. Programming with toolkits is suited to the notification-based programming paradigm. As we can see in the example, the button is created as a `PANEL_BUTTON` object (lines 23–26) and registers the appropriate callback routine for when the notifier receives a selection event for the button object. The button interaction object in the toolkit already has defined what actual user action is classified as the selection event, so the programmer need not worry about that when creating an instance of the button. The programmer can think of the event at a higher level of abstraction, that is as a selection event instead of as a release of the left mouse button.

In Chapter 7 we discussed the benefits of consistency and generalizability for an interactive system. One of the advantages of programming with toolkits is that they can enforce consistency in both input form and output form by providing similar behavior to a collection of widgets. For example, every button interaction object, within the same application program or between different ones, by default could have a behavior like the one described in Figure 8.8. All that is required is that the developers for the different applications use the same toolkit. This consistency of behavior for interaction objects is referred to as the *look and feel* of the toolkit. Style guides, which were described in the discussion on guidelines in Chapter 7, give additional hints to a programmer on how to preserve the look and feel of a given toolkit beyond that which is enforced by the default definition of the interaction objects.

Two features of interaction objects and toolkits make them amenable to an object-oriented approach to programming. First, they depend on being able to define a class of interaction objects which can then be invoked (or instantiated) many times within one application with only minor modifications to each instance. Secondly, building complex interaction objects is made easier by building up their definition based on existing simpler interaction objects. These notions of *instantiation* and *inheritance* are cornerstones of object-oriented programming. *Classes* are defined as templates for interaction objects. When an interaction object is created, it is declared as an instance of some predefined class. So, in the example `quit.c` program, `frame` is declared as an instance of the class `FRAME` (line 17), `panel` is declared as an instance of the class `PANEL` (line 22) and the button (no name) is declared as an instance of the class `PANEL_BUTTON` (line 23). Typically, a class template will provide default

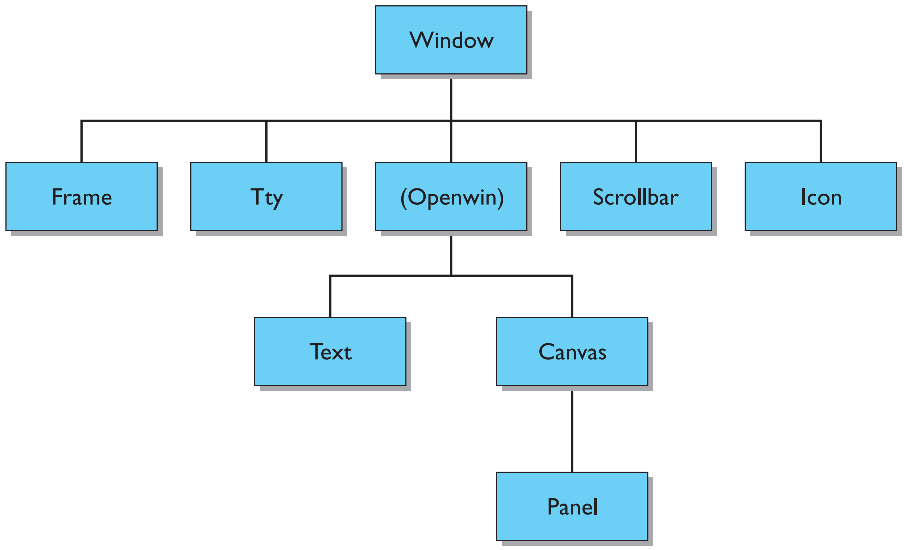


Figure 8.9 The single inheritance class hierarchy of the XView toolkit, after Heller [169], reproduced by permission of O'Reilly and Associates, Inc

values for various attributes. Some of those attributes can be altered in any one instance; they are sometimes distinguished as *instance attributes*.

In defining the classes of interaction objects themselves, new classes can be built which inherit features of one or other classes. In the simplest case, there is a strict *class hierarchy* in which each class inherits features of only one other class, its *parent class*. This simple form of inheritance is called *single inheritance* and is exhibited in the XView toolkit standard hierarchy for the window class in Figure 8.9. A more complicated class hierarchy would permit defining new classes which inherit from more than one parent class – called *multiple inheritance*.

DESIGN FOCUS

Java and AWT

The Java toolkit for developing windowed applications is called the Abstract Windowing Toolkit, AWT. It maps interface objects such as buttons, menus and dialog boxes onto corresponding Java classes. The programmer builds an interface either by using these classes directly or by *subclassing* them, that is specializing the behavior of the object in some way. This subclassing means that new interaction widgets can easily be added. The toolkit is notification based, but the mechanism has changed slightly between versions. In AWT 1.0 the programmer needs to subclass a button in order to specify its behavior when pressed. Since AWT 1.1 the programmer can use a method more like traditional callbacks, but based on registering special Java objects rather than functions.

We should point out that though most toolkits are structured in an object-oriented manner, that does not mean that the actual application programming language is object oriented. The example program `quit.c` was written in the C programming language, which is not an object-oriented language. It is best to think of object orientation as yet another programming paradigm which structures the way the programmer attacks the programming task without mandating a particular syntax or semantics for the programming language.

The programmer can tailor the behavior and appearance of an interaction object by setting the values of various instance attributes. These attributes must be set before the application program is compiled. In addition, some windowing systems allow various attributes of interaction objects to be altered without necessitating recompilation, though they may have to be set before the actual program is run. This tailorability is achieved via *resources* which can be accessed by the application program and change the compiled value of some attributes. For efficiency reasons, this tailorability is often limited to a small set of attributes for any given class.

Worked exercise

Scrolling is an effective means of browsing through a document in a window that is too small to show the whole document. Compare the different interactive behavior of the following two interaction objects to implement scrolling:

- *A scrollbar is attached to the side of the window with arrows at the top and bottom. When the mouse is positioned over the arrow at the top of the screen (which points up), the window frame is moved upwards to reveal a part of the document above/before what is currently viewed. When the bottom arrow is selected, the frame moves down to reveal the document below/after the current view.*
- *The document is contained in a textual interaction object. Pressing the mouse button in the text object allows you to drag the document within the window boundaries. You drag up to browse down in the document and you drag down to browse up.*

The difference between the two situations can be characterized by noticing that, in the first case, the user is actually manipulating the window (moving it up or down to reveal the contents of the document), whereas, in the second case, the user is manipulating the document (pushing it up or down to reveal its contents through the windows). What usability principles would you use to justify one method over the other (also consider the case when you want to scroll from side to side as well as up and down)? What implementation considerations are important?

Answer

There are many usability principles that can be brought to bear on an examination of scrolling principles. For example:

Observability The whole reason why scrolling is used is because there is too much information to present all at once. Providing a means of viewing document contents without changing the contents increases the observability of the system. Scrollbars also increase observability because they help to indicate the wider context of the information which is currently visible, typically by showing where the window of information fits within the whole document. However, observability does not address the particular design options put forth here.

Predictability The value of a scrolling mechanism lies in the user being able to know where a particular scrolling action will lead in the document. The use of arrows on the scrollbar is to help the user predict the effect of the scrolling operation. If an arrow points up, the question is whether that indicates the direction the window is being moved (the first case) or the direction the actual text would have to move (the second case). The empirical question here is: to what object do users associate the arrow – the text or the text window? The arrow of the scrollbar is more closely connected to the boundary of a text window, so the more usual interpretation would be to have it indicate the direction of the window movement.

Synthesizability You might think that it does not matter which object the user associates to the arrow. He will just have to learn the mapping and live with it. In this case, how easy is it to learn the mapping, that is can the user synthesize the meaning of the scrolling actions from changes made at the display? Usually, the movement of a box within the scrollbar itself will indicate the result of a scrolling operation.

Familiarity/guessability It would be an interesting experiment to see whether there was a difference in the performance of new users for the different scrolling mechanisms. This might be the subject of a more extended exercise.

Task conformance There are some implementation limitations for these scrolling mechanisms (see below). In light of these limitations, does the particular scrolling task prefer one over the other? In considering this principle, we need to know what kinds of scrolling activity will be necessary. Is the document a long text that will be browsed from end to end, or is it possibly a map or a picture which is only slightly larger than the actual screen so scrolling will only be done in small increments?

Some implementation considerations:

- What scroll mechanisms does a toolkit provide? Is it easy to access the two options discussed above within the same toolkit?
- In the case of the second scrolling option, are there enough keys on the mouse to allow this operation without interfering with other important mouse operations, such as arbitrarily moving the insertion point or selecting a portion of text or selecting a graphical item?
- In the second option, the user places the mouse on a specific location within the window, and gestures to dictate the movement of the underlying document. What kind of behavior is expected when the mouse hits the boundary of the window? Is the scrolling limited in this case to steps bounded in size by the size of the window, so that scrolling between two distant points requires many separate smaller scrolling actions?

8.5 USER INTERFACE MANAGEMENT SYSTEMS

Despite the availability of toolkits and the valuable abstraction they provide programmers, there are still significant hurdles to overcome in the specification, design and implementation of interactive systems. Toolkits provide only a limited range of interaction objects, limiting the kinds of interactive behavior allowed between user and system. Toolkits are expensive to create and are still very difficult to use by non-programmers. Even experienced programmers will have difficulty using them to produce an interface that is predictably usable. There is a need for additional support for programmers in the design and use of toolkits to overcome their deficiencies. Also, none of the programming mechanisms we have discussed so far in this chapter is appropriate for non-expert programmers, so we still have a long way to go towards the goal of opening up interactive system implementation to those whose main concerns are with HCI and not programming.

The set of programming and design techniques which are supposed to add another level of services for interactive system design beyond the toolkit level are *user interface management systems*, or *UIMS* for short. The term UIMS is used quite widely in both industrial and academic circles and has come to represent a variety of topics. The main concerns of a UIMS, for our purposes, are:

- a conceptual architecture for the structure of an interactive system which concentrates on a separation between application semantics and presentation;
- techniques for implementing a separated application and presentation whilst preserving the intended connection between them;
- support techniques for managing, implementing and evaluating a run-time interactive environment.

We should acknowledge that some people feel that the term UIMS is inappropriate for all of the above tasks, preferring the term *user interface development systems*, or *UIDS*, to distinguish support tools which address many of the design activities that precede the management of the run-time system.

8.5.1 UIMS as a conceptual architecture

A major issue in this area of research is one of *separation* between the semantics of the application and the interface provided for the user to make use of that semantics. There are many good arguments to support this separation of concerns:

Portability To allow the same application to be used on different systems it is best to consider its development separate from its device-dependent interface.

Reusability Separation increases the likelihood that components can be reused in order to cut development costs.

Multiple interfaces To enhance the interactive flexibility of an application, several different interfaces can be developed to access the same functionality.

Customization The user interface can be customized by both the designer and the user to increase its effectiveness without having to alter the underlying application.

Once we allow for a separation between application and presentation, we must consider how those two partners communicate. This role of communication is referred to as *dialog control*. Conceptually, this provides us with the three major components of an interactive system: the application, the presentation and the dialog control. In terms of the actual implementation, this separation may not be so clear.

In Section 8.3, we described the two basic approaches to programming the application within an interactive system. In the read–evaluation loop, the control of the dialog is *internal* to the application. The application calls interface procedures when input or output is required. In notification-based programming, the dialog control resides *external* to the application. When the user performs some input action, the notifier then invokes the correct application procedure to handle the event. Most UIMS fall into this class of external dialog control systems, since they promote, to a greater extent, the separation between presentation and application. They do not, however, all use the technique of callbacks as was demonstrated in Section 8.4 for the use of toolkits.

The first acknowledged instance of a development system that supported this application–presentation separation was in 1968 with Newman's Reaction Handler. The term UIMS was coined by Kasik in 1982 [196a] after some preliminary research on how graphical input could be used to broaden the scope of HCI. The first conceptual architecture of what constituted a UIMS was formulated at a workshop in 1985 at Seeheim, Germany [285]. The logical components of a UIMS were identified as:

Presentation The component responsible for the appearance of the interface, including what output and input is available to the user.

Dialog control The component which regulates the communication between the presentation and the application.

Application interface The view of the application semantics that is provided as the interface.

Figure 8.10 presents a graphical interpretation of the Seeheim model. We have included both application and user in Figure 8.10 to place the UIMS model more in the context of the interactive system (though you could argue that we have not provided enough of that context by mentioning only a single user and a single application). The application and the user are not explicit in the Seeheim model

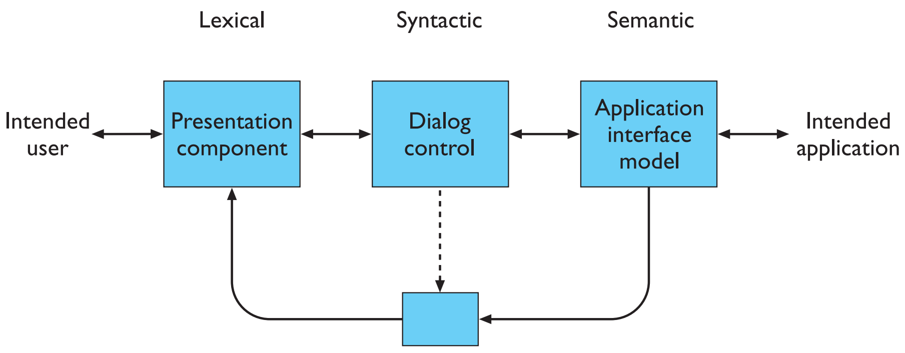


Figure 8.10 The Seeheim model of the logical components of a UIMS

because it was intended only to model the components of a UIMS and not the entire interactive system. By not making the application explicit in the model, external dialog control must have been assumed. From a programmer's perspective, the Seeheim model fits in nicely with the distinction between the classic lexical, syntactic and semantic layers of a computer system, familiar from compiler design.

One of the main problems with the Seeheim model is that, whereas it served well as a post hoc rationalization of how a UIMS was built up to 1985, it did not provide any real direction for how future UIMS should be structured. A case in point can be seen in the inclusion of the lowest box in Figure 8.10, which was intended to show that for efficiency reasons it would be possible to bypass an explicit dialog control component so that the application could provide greater application semantic feedback. There is no need for such a box in a conceptual architecture of the logical components. It is there because its creators did not separate logical concerns from implementation concerns.

Semantic feedback

One of the most ill-understood elements of the *Seeheim model* is the lower box: the *bypass* or *switch*. This is there to allow rapid semantic feedback. Examples of semantic feedback include freehand drawing and the highlighting of the trash bin on the Apple Macintosh when a file is dragged over it. As with all notions of levels in interface design, the definition of semantic feedback is not sharp, but it corresponds to those situations where it is impractical or impossible to use dialog-level abstractions to map application structures to screen representations.

The box represents the fact that in such circumstances the application component needs to address the presentation component directly, often to achieve suitable performance. It thus bypasses the dialog component. However, the box has an arrow from the dialog component which represents not a data flow, but control. Although the dialog does not mediate the presentation of information, it does control when and where the application is allowed to access the presentation; hence the alternative name of switch.

In graphical and WIMP-based systems the Seeheim components seem restrictive as single entities, and partly in response to this a later workshop developed the Arch–Slinky model [354]. This has more layers than the Seeheim model and, more importantly, recognizes that the mapping of these layers to components of a system may be more fluid than Seeheim suggests.

Another concern not addressed by the Seeheim model is how to build large and complex interactive systems from smaller components. We have seen that object-based toolkits are amenable to such a building blocks approach, and several other conceptual architectures for interactive system development have been proposed to take advantage of this. One of the earliest was the *model–view–controller* paradigm – MVC for short – suggested in the *Smalltalk* programming environment [233, 203, 212]. Smalltalk was one of the earliest successful object-oriented programming systems whose main feature was the ability to build new interactive systems based on

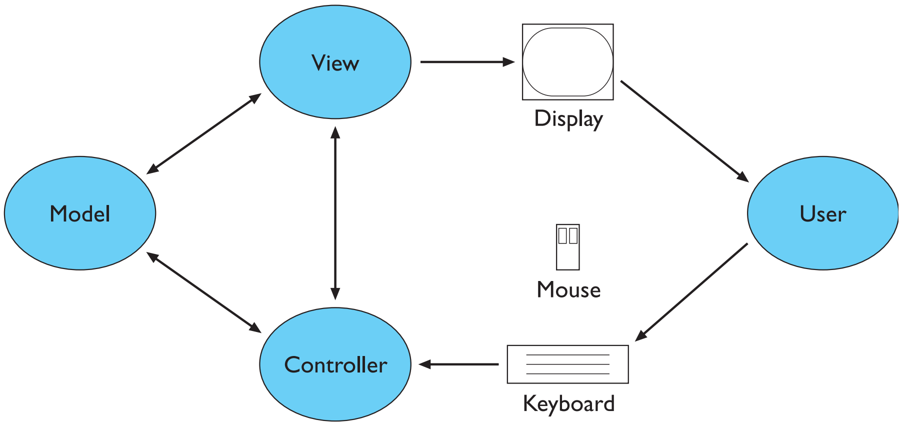


Figure 8.11 The model–view–controller triad in Smalltalk

existing ones. Within Smalltalk, the link between application semantics and presentation can be built up in units by means of the MVC triad. The model represents the application semantics; the view manages the graphical and/or textual output of the application; and the controller manages the input (see Figure 8.11).

The basic behavior of models, views and controllers has been embodied in general Smalltalk object classes, which can be inherited by instances and suitably modified. Smalltalk, like many other window toolkits, prescribes its own look and feel on input and output, so the generic view and controller classes (called `View` and `Controller`, respectively) do not need much modification after instantiation. Models, on the other hand, are very general because they must be used to portray any possible application semantics. A single model can be associated with several MVC triads, so that the same piece of application semantics can be represented by different input–output techniques. Each view–controller pair is associated to only one model.

Another so-called *multi-agent* architecture for interactive systems is the *presentation–abstraction–control* PAC model suggested by Coutaz [79]. PAC is based on a collection of triads also: with application semantics represented by the abstraction component; input and output combined in one presentation component; and an explicit control component to manage the dialog and correspondence between application and presentation (see Figure 8.12). There are three important differences between PAC and MVC. First, PAC groups input and output together, whereas MVC separates them. Secondly, PAC provides an explicit component whose duty it is to see that abstraction and presentation are kept consistent with each other, whereas MVC does not assign this important task to any one component, leaving it to the programmer/designer to determine where that chore resides. Finally, PAC is not linked to any programming environment, though it is certainly conducive to an object-oriented approach. It is probably because of this last difference that PAC could so easily isolate the control component; PAC is more of a conceptual architecture than MVC because it is less implementation dependent.

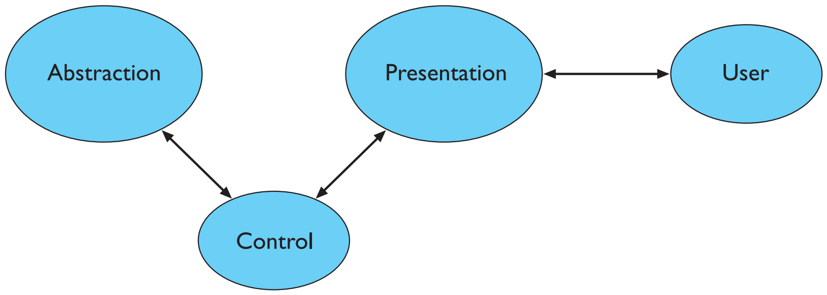


Figure 8.12 The presentation–abstraction–control model of Coutaz

8.5.2 Implementation considerations

We have made a point of distinguishing a conceptual architecture from any implementation considerations. It is, however, important to determine how components in a conceptual architecture can be realized. Implementations based on the Seeheim model must determine how the separate components of presentation, dialog controller and application interface are realized. Window systems and toolkits provide the separation between application and presentation. The use of callback procedures in notification-based programming is one way to implement the application interface as a notifier. In the standard X toolkit, these callbacks are directional as it is the duty of the application to register itself with the notifier. In MVC, callback procedures are also used for communication between a view or controller and its associated model, but this time it is the duty of the presentation (the view or controller) to register itself with the application (the model). Communication from the model to either view or controller, or between a view and a controller, occurs by the normal use of method calls used in object-oriented programming. Neither of these provides a means of separately managing the dialog.

Myers has outlined the various implementation techniques used to specify the dialog controller separately. Many of these will be discussed in Chapter 16 where we explicitly deal with dialog notations. Some of the techniques that have been used in dialog modeling in UIMS are listed here.

Menu networks The communication between application and presentation is modeled as a network of menus and submenus. To control the dialog, the programmer must simply encode the levels of menus and the connections between one menu and the next submenu or an action. The menu is used to embody all possible user inputs at any one point in time. Links between menu items and the next displayed menu model the application response to previous input. A menu does not have to be a linear list of textual actions. The menu can be represented as graphical items or buttons that the user can select with a pointing device. Clicking on one button moves the dialog to the next screen of objects. In this way, a system like HyperCard can be considered a menu network.

Grammar notations The dialog between application and presentation can be treated as a grammar of actions and responses, and, therefore, described by means of a formal context-free grammar notation, such as BNF (Backus–Naur form). These are good for describing command-based interfaces, but are not so good for more graphically-based interaction techniques. It is also not clear from a formal grammar what directionality is associated to each event in the grammar; that is, whether an event is initiated by the user or by the application. Therefore, it is difficult to model communication of values across the dialog controller, and that is necessary to maintain any semantic feedback from application to presentation.

State transition diagrams State transition diagrams can be used as a graphical means of expressing dialog. Many variants on state transition diagrams will be discussed in Chapter 16. The difficulty with these notations lies in linking dialog events with corresponding presentation or application events. Also, it is not clear how communication between application and presentation is represented.

Event languages Event languages are similar to grammar notations, except that they can be modified to express directionality and support some semantic feedback. Event languages are good for describing localized input–output behavior in terms of production rules. A production rule is activated when input is received and it results in some output responses. This control of the input–output relationship comes at a price. It is now more difficult to model the overall flow of the dialog.

Declarative languages All of the above techniques (except for menu networks) are poor for describing the correspondence between application and presentation because they are unable to describe effectively how information flows between the two. They only view the dialog as a sequence of events that occur between two communicating partners. A declarative approach concentrates more on describing how presentation and application are related. This relationship can be modeled as a shared database of values that both presentation and application can access. Declarative languages, therefore, describe what should result from the communication between application and presentation, not how it should happen in terms of event sequencing.

Constraints Constraints systems are a special subset of declarative languages. *Constraints* can be used to make explicit the connection between independent information of the presentation and the application. Implicit in the control component of the PAC model is this notion of constraint between values of the application and values of the presentation. Hill has proposed the *abstraction–link–view*, or ALV (pronounced 'AL-vee'), which makes the same distinctions as PAC [172]. However, Hill suggests an implementation of the communication between abstraction and view by means of the link component as a collection of two-way constraints between abstraction and view. Constraints embody dependencies between different values that must always be maintained. For instance, an intelligent piggy bank might display the value of its contents; there is the constraint that the value displayed to the outside observer of the piggy bank is the same as the value of money inside it. By using constraints, the link component is described separately from the abstraction and view. Hence, describing the link in terms of constraints is a way of achieving an independent description of the dialog controller.

Graphical specification These techniques allow the dialog specification to be programmed graphically in terms of the presentation language itself. This technique can be referred to as *programming by demonstration* since the programmer is building up the interaction dialog directly in terms of the actual graphical interaction objects that the user will see, instead of indirectly by means of some textual specification language that must still be linked with the presentation objects. The major advantage of this graphical technique is that it opens up the dialog specification to the non-programmer, which is a very significant contribution.

Ultimately, the programmer would want access to a variety of these techniques in any one UIMS. For example, the Myers Garnet system combines a declarative constraints language with a graphical specification technique. There is an intriguing trend we should note as we proceed away from internal control of dialog in the application itself to external control in an independent dialog component to *presentation control* in the graphical specification languages. When the dialog is specified internal to the application, then it must know about presentation issues, which make the application less generic. External control is about specifying the dialog independent of the application or presentation. One of the problems with such an independent description is that the intended link between application and presentation is impossible to describe without some information about each, so a good deal of information of each must be represented, which may be both inefficient and cumbersome. Presentation control describes the dialog in the language in terms of the objects the user can see at the interface. Whereas this might provide a simple means of producing a dialog specification and be more amenable to non-programmers, it is also restrictive because the graphical language of a modern workstation is nowhere near as expressive as programming languages.

In summary, components of a UIMS which allow the description of the application separate from the presentation are advantageous from a software engineering perspective, but there has not yet been conclusive proof that they are as desirable in designing for usability. There is currently a struggle between difficult-to-use but powerful techniques for describing both the communication and the correspondence between application and presentation and simple-to-use but limited techniques. Programmers will probably always opt for powerful techniques that provide the most flexibility. Non-programmers will opt for simplicity despite the lack of expressiveness.

8.6 SUMMARY

In this chapter, we have concentrated on describing the programming support tools that are available for implementing interactive systems. We began with a description of windowing systems, which are the foundation of modern WIMP interfaces.

Window systems provide only the crudest level of abstraction for the programmer, allowing her to gain device independence and multiple application control. They do not, however, provide a means of separating the control of presentation and application dialog. We described two paradigms for interactive programming, and saw that these relate to two means of controlling that dialog – either internal to the application by means of a read–evaluation loop or external to the application by means of notification-based programming. Toolkits used with particular windowing systems add another level of abstraction by combining input and output behaviors to provide the programmer with access to interaction objects from which to build the components of the interactive system. Toolkits are amenable to external dialog control by means of callback procedures within the application. Other dialog control techniques are provided with yet another level of abstraction in interactive system development: user interface management systems. UIMS provide a conceptual architecture for dividing up the relationship between application and presentation, and various techniques were described to implement the logical components of a UIMS. An interesting additional means of dialog control can be seen to emerge in the use of graphical specification languages which move dialog control all the way across the spectrum to reside entirely within the presentation language. This presentation control opens up interactive programming to the non-expert programmer, but at the cost of a loss of expressiveness.

EXERCISES

8.1 In contrasting the read–evaluation loop and the notification-based paradigm for interactive programs, construction of a pre-emptive dialog was discussed. How would a programmer describe a pre-emptive dialog by purely graphical means? (**Hint:** Refer to the discussion in Section 8.5 concerning the shift from external and independent dialog management to presentation control of the dialog.)

8.2 Look ahead to the example of the state transition diagram for font characteristics presented in Chapter 16 (Section 16.3.3). Compare different interaction objects that could implement this kind of dialog. Use examples from existing toolkits (pull-down menus or dialog boxes) or create a novel interaction object.

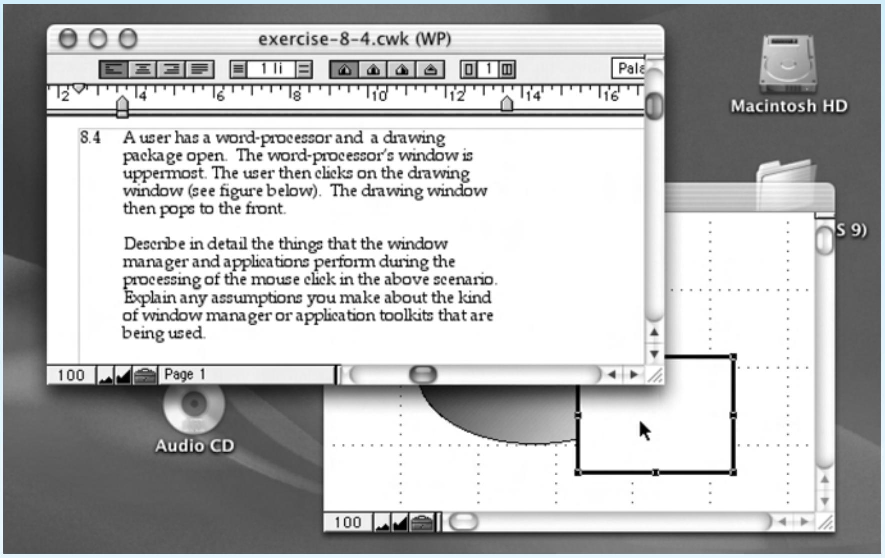
8.3 This exercise is based on the nuclear reactor scenario on the book website at: /e3/scenario/nuclear/

(a) In the Seeheim model: treating the Application Interface model and Application together, there are three main layers:
(i) presentation/lexical
(ii) dialog/syntactic
(iii) application/semantic.
For each of these three layers, list at least two different items of the description of the nuclear reactor control panel that are relevant to the level (that is, at least six items in total, two for each level).

(b) There are no items in the description that relate to the switch (rapid feedback) part of the Seeheim model. Why do you think this is?

8.4 A user has a word processor and a drawing package open. The word processor's window is uppermost. The user then clicks on the drawing window (see figure below). The drawing window pops to the front.

Describe in detail the things that the window manager and applications perform during the processing of the mouse click in the above scenario. Explain any assumptions you make about the kind of window manager or application toolkits that are being used.



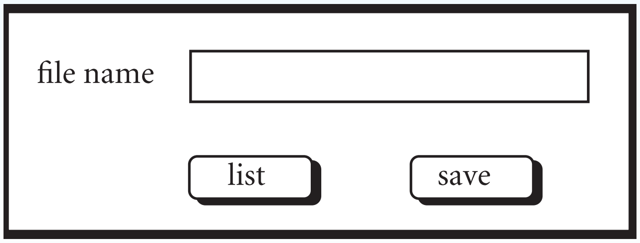
8.4 A user has a word-processor and a drawing package open. The word-processor's window is uppermost. The user then clicks on the drawing window (see figure below). The drawing window then pops to the front.

Describe in detail the things that the window manager and applications perform during the processing of the mouse click in the above scenario. Explain any assumptions you make about the kind of window manager or application toolkits that are being used.

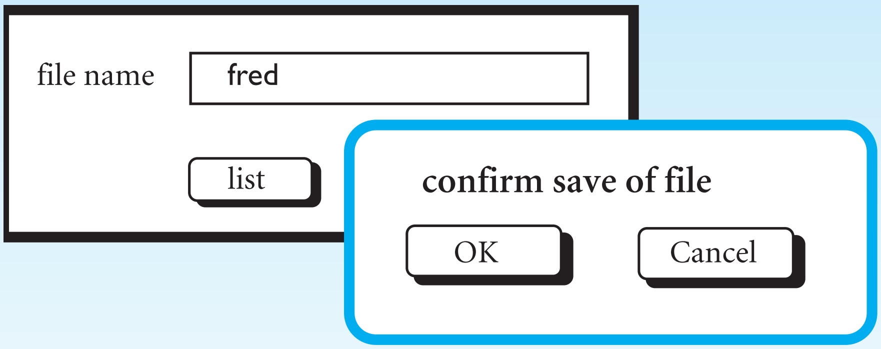
Screen shot reprinted by permission from Apple Computer, Inc.

8.5 A designer described the following interface for a save operation.

The users initially see a screen with a box where they can type the file name (see Screen 1). The screen also has a 'list' button that they can use to obtain a listing of all the files in the current directory (folder). This list appears in a different window. When the user clicks the save button, the system presents a dialog box to ask the user to confirm the save (see Screen 2).



Screen 1



Screen 2

Two programmers independently coded the interface using two different window managers. Programmer A used an event-loop style of program whereas programmer B used a notifier (call-back) style.

(a) Sketch out the general structure of each program.
(b) Highlight any potential interface problems you expect from each programmer and how they could attempt to correct them.

RECOMMENDED READING

L. Bass and J. Coutaz, *Developing Software for the User Interface*, Addison-Wesley, 1991.
This is dedicated to the issues we discuss in this chapter, along with general issues about software engineering for interactive systems. Full of programming examples and a detailed discussion of the Serpent UIMS.

B. A. Myers, *Creating User Interfaces by Demonstration*, Academic Press, 1988.
Myers' work on the Peridot system is summarized in this book. Peridot was the precursor to the Garnet system. Readers interested in learning more about Garnet should consult the November 1990 issue of the journal *IEEE Computer* for an excellent introductory overview.

D. Olsen, *User Interface Management Systems: Models and Algorithms*, Morgan Kaufmann, 1992.
Serious interactive system programmers who want to learn more details about the workings of a wide variety of UIMS should consult this book, written by a very respected member of the UIMS community.

D. Hix, Generations of user-interface management systems, *IEEE Software*, Vol. 7, No. 5, pp. 77–87, September 1990.
A good introductory overview of the advances in UIMS technology and the future possibilities.

B. A. Myers, User-interface tools: introduction and survey, *IEEE Software*, Vol. 6, No. 1, pp. 47–61, January 1989.
As well as providing a review of user interface tools, this article provides a good set of references into the relevant literature.

G. F. Coulouris and H. W. Thimbleby, *HyperProgramming*, Addison-Wesley, 1993.
An introduction to programming HyperCard. Use of the programming facilities of HyperTalk allows one to prototype substantial parts of the functionality as well as the surface features of an interface.

P. H. Winston and S. Narasimhan, *On to Java*, 3rd edition, Addison-Wesley, 2001.
Java is another good way to get into user interface programming, as examples can easily be embedded into web pages as applets. This is a good first book on Java using the construction of an applet in AWT as its motivating example. For clear reference books on aspects of the language, look at the O'Reilly series.

C. Gram and G. Cockton, editors, *Design Principles for Interactive Software*, Chapman and Hall, 1996.
Produced by IFIP Working Group 2.7 (User Interface Engineering). This critically discusses several user interface architectures and looks at the way architecture can help or hinder the pursuit of principles similar to those in Chapter 7.

9

EVALUATION TECHNIQUES

OVERVIEW

- Evaluation tests the usability, functionality and acceptability of an interactive system.
- Evaluation may take place:
  - in the laboratory
  - in the field.
- Some approaches are based on expert evaluation:
  - analytic methods
  - review methods
  - model-based methods.
- Some approaches involve users:
  - experimental methods
  - observational methods
  - query methods.
- An evaluation method must be chosen carefully and must be suitable for the job.

9.1 WHAT IS EVALUATION?

In previous chapters we have discussed a design process to support the design of usable interactive systems. However, even if such a process is used, we still need to assess our designs and test our systems to ensure that they actually behave as we expect and meet user requirements. This is the role of evaluation.

Evaluation should not be thought of as a single phase in the design process (still less as an activity tacked on the end of the process if time permits). Ideally, evaluation should occur throughout the design life cycle, with the results of the evaluation feeding back into modifications to the design. Clearly, it is not usually possible to perform extensive experimental testing continuously throughout the design, but analytic and informal techniques can and should be used. In this respect, there is a close link between evaluation and the principles and prototyping techniques we have already discussed – such techniques help to ensure that the design is assessed continually. This has the advantage that problems can be ironed out before considerable effort and resources have been expended on the implementation itself: it is much easier to change a design in the early stages of development than in the later stages. We can make a broad distinction between evaluation by the designer or a usability expert, without direct involvement by users, and evaluation that studies actual use of the system. The former is particularly useful for assessing early designs and prototypes; the latter normally requires a working prototype or implementation. However, this is a broad distinction and, in practice, the user may be involved in assessing early design ideas (for example, through focus groups), and expert-based analysis can be performed on completed systems, as a cheap and quick usability assessment. We will consider evaluation techniques under two broad headings: expert analysis and user participation.

Before looking at specific techniques, however, we will consider why we do evaluation and what we are trying to achieve.

9.2 GOALS OF EVALUATION

Evaluation has three main goals: to assess the extent and accessibility of the system's functionality, to assess users' experience of the interaction, and to identify any specific problems with the system.

The system's functionality is important in that it must accord with the user's requirements. In other words, the design of the system should enable users to perform their intended tasks more easily. This includes not only making the appropriate functionality available within the system, but making it clearly reachable by the user in terms of the actions that the user needs to take to perform the task. It also involves matching the use of the system to the user's expectations of the task. For example, if a filing clerk is used to retrieving a customer's file by the postal address, the same capability (at least) should be provided in the computerized file system. Evaluation at this level may also include measuring the user's performance with the system, to assess the effectiveness of the system in supporting the task.

In addition to evaluating the system design in terms of its functional capabilities, it is important to assess the user's experience of the interaction and its impact upon him. This includes considering aspects such as how easy the system is to learn, its usability and the user's satisfaction with it. It may also include his enjoyment and emotional response, particularly in the case of systems that are aimed at leisure or entertainment. It is important to identify areas of the design that overload the user in some way, perhaps by requiring an excessive amount of information to be remembered, for example. A fuller classification of principles that can be used as evaluation criteria is provided in Chapter 7. Much evaluation is aimed at measuring features such as these.

The final goal of evaluation is to identify specific problems with the design. These may be aspects of the design which, when used in their intended context, cause unexpected results, or confusion amongst users. This is, of course, related to both the functionality and usability of the design (depending on the cause of the problem). However, it is specifically concerned with identifying trouble-spots which can then be rectified.

9.3 EVALUATION THROUGH EXPERT ANALYSIS

As we have noted, evaluation should occur throughout the design process. In particular, the first evaluation of a system should ideally be performed before any implementation work has started. If the design itself can be evaluated, expensive mistakes can be avoided, since the design can be altered prior to any major resource commitments. Typically, the later in the design process that an error is discovered, the more costly it is to put right and, therefore, the less likely it is to be rectified. However, it can be expensive to carry out user testing at regular intervals during the design process, and it can be difficult to get an accurate assessment of the experience of interaction from incomplete designs and prototypes. Consequently, a number of methods have been proposed to evaluate interactive systems through expert analysis. These depend upon the designer, or a human factors expert, taking the design and assessing the impact that it will have upon a typical user. The basic intention is to identify any areas that are likely to cause difficulties because they violate known cognitive principles, or ignore accepted empirical results. These methods can be used at any stage in the development process from a design specification, through storyboards and prototypes, to full implementations, making them flexible evaluation approaches. They are also relatively cheap, since they do not require user involvement. However, they do not assess actual use of the system, only whether or not a system upholds accepted usability principles.

We will consider four approaches to expert analysis: cognitive walkthrough, heuristic evaluation, the use of models and use of previous work.

9.3.1 Cognitive walkthrough

Cognitive walkthrough was originally proposed and later revised by Polson and colleagues [294, 376] as an attempt to introduce psychological theory into the informal and subjective walkthrough technique.

The origin of the cognitive walkthrough approach to evaluation is the code walkthrough familiar in software engineering. Walkthroughs require a detailed review of a sequence of actions. In the code walkthrough, the sequence represents a segment of the program code that is stepped through by the reviewers to check certain characteristics (for example, that coding style is adhered to, conventions for spelling variables versus procedure calls, and to check that system-wide invariants are not violated). In the cognitive walkthrough, the sequence of actions refers to the steps that an interface will require a user to perform in order to accomplish some known task. The evaluators then 'step through' that action sequence to check it for potential usability problems. Usually, the main focus of the cognitive walkthrough is to establish how easy a system is to learn. More specifically, the focus is on learning through exploration. Experience shows that many users prefer to learn how to use a system by exploring its functionality hands on, and not after sufficient training or examination of a user's manual. So the checks that are made during the walkthrough ask questions that address this exploratory learning. To do this, the evaluators go through each step in the task and provide a 'story' about why that step is or is not good for a new user. To do a walkthrough (the term walkthrough from now on refers to the cognitive walkthrough, and not to any other kind of walkthrough), you need four things:

1. A specification or prototype of the system. It doesn't have to be complete, but it should be fairly detailed. Details such as the location and wording for a menu can make a big difference.
2. A description of the task the user is to perform on the system. This should be a representative task that most users will want to do.
3. A complete, written list of the actions needed to complete the task with the proposed system.
4. An indication of who the users are and what kind of experience and knowledge the evaluators can assume about them.

Given this information, the evaluators step through the action sequence (identified in item 3 above) to critique the system and tell a believable story about its usability. To do this, for each action, the evaluators try to answer the following four questions for each step in the action sequence.

1. **Is the effect of the action the same as the user's goal at that point?** Each user action will have a specific effect within the system. Is this effect the same as what the user is trying to achieve at this point? For example, if the effect of the action is to save a document, is 'saving a document' what the user wants to do?
2. **Will users see that the action is available?** Will users see the button or menu item, for example, that is used to produce the action? This is *not* asking whether they will recognize that the button is the one they want. This is merely asking whether

it is visible to them at the time when they will need to use it. Instances where the answer to this question might be 'no' are, for example, where a VCR remote control has a covered panel of buttons or where a menu item is hidden away in a submenu.

3. **Once users have found the correct action, will they know it is the one they need?** This complements the previous question. It is one thing for a button or menu item to be visible, but will the user recognize that it is the one he is looking for to complete his task? Where the previous question was about the visibility of the action, this one is about whether its meaning and effect is clear.
4. **After the action is taken, will users understand the feedback they get?** If you now assume that the user did manage to achieve the correct action, will he know that he has done so? Will the feedback given be sufficient confirmation of what has actually happened? This is the completion of the execution–evaluation interaction cycle (see Chapter 3). In order to determine if they have accomplished their goal, users need appropriate feedback.

It is vital to document the cognitive walkthrough to keep a record of what is good and what needs improvement in the design. It is therefore a good idea to produce some standard evaluation forms for the walkthrough. The cover form would list the information in items 1–4 in our first list above, as well as identifying the date and time of the walkthrough and the names of the evaluators. Then for each action (from item 3 on the cover form), a separate standard form is filled out that answers each of the four questions in our second list above. Any negative answer for any of the questions for any particular action should be documented on a separate usability problem report sheet. This problem report sheet should indicate the system being built (the version, if necessary), the date, the evaluators and a detailed description of the usability problem. It is also useful to indicate the severity of the problem, that is whether the evaluators think this problem will occur often, and how serious it will be for the users. This information will help the designers to decide priorities for correcting the design, since it is not always possible to fix every problem.

Example: programming a video recorder by remote control

We can illustrate how the walkthrough method works using a simple example. Imagine we are designing a remote control for a video recorder (VCR) and are interested in the task of programming the VCR to do timed recordings. Our initial design is shown in Figure 9.1. The picture on the left illustrates the handset in normal use, the picture on the right after the timed record button has been pressed. The VCR allows the user to program up to three timed recordings in different 'streams'. The next available stream number is automatically assigned. We want to know whether our design supports the user's task. We begin by identifying a representative task.

Program the video to time-record a program starting at 18.00 and finishing at 19.15 on channel 4 on 24 February 2005.

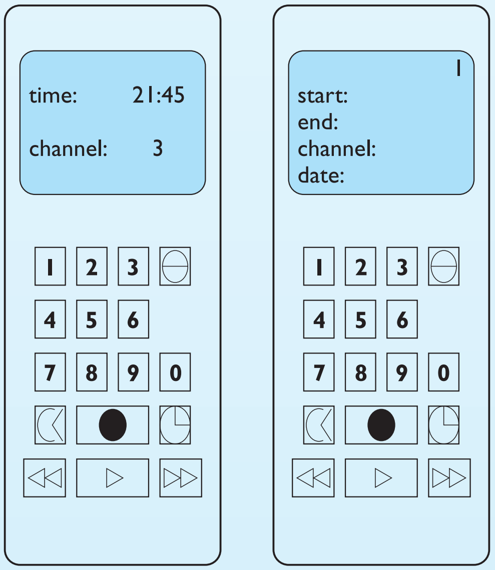


Figure 9.1 An initial remote control design

We will assume that the user is familiar with VCRs but not with this particular design.

The next step in the walkthrough is to identify the action sequence for this task. We specify this in terms of the user's action (UA) and the system's display or response (SD). The initial display is as the left-hand picture in Figure 9.1.

UA 1: Press the 'timed record' button
SD 1: Display moves to timer mode. Flashing cursor appears after 'start:'
UA 2: Press digits 1 8 0 0
SD 2: Each digit is displayed as typed and flashing cursor moves to next position
UA 3: Press the 'timed record' button
SD 3: Flashing cursor moves to 'end:'
UA 4: Press digits 1 9 1 5
SD 4: Each digit is displayed as typed and flashing cursor moves to next position
UA 5: Press the 'timed record' button
SD 5: Flashing cursor moves to 'channel:'
UA 6: Press digit 4
SD 6: Digit is displayed as typed and flashing cursor moves to next position
UA 7: Press the 'timed record' button
SD 7: Flashing cursor moves to 'date:'
UA 8: Press digits 2 4 0 2 0 5
SD 8: Each digit is displayed as typed and flashing cursor moves to next position
UA 9: Press the 'timed record' button
SD 9: Stream number in top right-hand corner of display flashes
UA 10: Press the 'transmit' button
SD 10: Details are transmitted to video player and display returns to normal mode

Having determined our action list we are in a position to proceed with the walkthrough. For each action (1–10) we must answer the four questions and tell a story about the usability of the system. Beginning with UA 1:

UA 1: Press the 'timed record' button
Question 1: Is the effect of the action the same as the user's goal at that point?
The timed record button initiates timer programming. It is reasonable to assume that a user familiar with VCRs would be trying to do this as his first goal.
Question 2: Will users see that the action is available?
The 'timed record' button is visible on the remote control.
Question 3: Once users have found the correct action, will they know it is the one they need?
It is not clear which button is the 'timed record' button. The icon of a clock (fourth button down on the right) is a possible candidate but this could be interpreted as a button to change the time. Other possible candidates might be the fourth button down on the left or the filled circle (associated with record). In fact, the icon of the clock is the correct choice but it is quite possible that the user would fail at this point. This identifies a potential usability problem.
Question 4: After the action is taken, will users understand the feedback they get?
Once the action is taken the display changes to the timed record mode and shows familiar headings (start, end, channel, date). It is reasonable to assume that the user would recognize these as indicating successful completion of the first action.

So we find we have a potential usability problem relating to the icon used on the 'timed record' button. We would now have to establish whether our target user group could correctly distinguish this icon from others on the remote.

The analysis proceeds in this fashion, with a walkthrough form completed for each action. We will leave the rest of the walkthrough for you to complete as an exercise. What other usability problems can you identify with this design?

9.3.2 Heuristic evaluation

A heuristic is a guideline or general principle or rule of thumb that can guide a design decision or be used to critique a decision that has already been made. *Heuristic evaluation*, developed by Jakob Nielsen and Rolf Molich, is a method for structuring the critique of a system using a set of relatively simple and general heuristics. Heuristic evaluation can be performed on a design specification so it is useful for evaluating early design. But it can also be used on prototypes, storyboards and fully functioning systems. It is therefore a flexible, relatively cheap approach. Hence it is often considered a *discount usability* technique.

The general idea behind heuristic evaluation is that several evaluators independently critique a system to come up with potential usability problems. It is important that there be several of these evaluators and that the evaluations be done independently. Nielsen's experience indicates that between three and five evaluators is sufficient, with five usually resulting in about 75% of the overall usability problems being discovered.

To aid the evaluators in discovering usability problems, a set of 10 heuristics are provided. The heuristics are related to *principles* and *guidelines* (see Chapter 7). These can be supplemented where required by heuristics that are specific to the particular domain. So, for example, if the system is for synchronous group communication, one might add ‘awareness of other users’ as a heuristic. Although Nielsen recommends the use of these 10 as providing the most effective coverage of the most common usability problems, other rules, such as those discussed in Chapter 7, could also be used.

Each evaluator assesses the system and notes violations of any of these heuristics that would indicate a potential usability problem. The evaluator also assesses the severity of each usability problem, based on four factors: how common is the problem, how easy is it for the user to overcome, will it be a one-off problem or a persistent one, and how seriously will the problem be perceived? These can be combined into an overall severity rating on a scale of 0–4:

0 = I don’t agree that this is a usability problem at all
1 = Cosmetic problem only: need not be fixed unless extra time is available on project
2 = Minor usability problem: fixing this should be given low priority
3 = Major usability problem: important to fix, so should be given high priority
4 = Usability catastrophe: imperative to fix this before product can be released (Nielsen)

Nielsen’s ten heuristics are:

1. **Visibility of system status** Always keep users informed about what is going on, through appropriate feedback within reasonable time. For example, if a system operation will take some time, give an indication of how long and how much is complete.
2. **Match between system and the real world** The system should speak the user’s language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in natural and logical order.
3. **User control and freedom** Users often choose system functions by mistake and need a clearly marked ‘emergency exit’ to leave the unwanted state without having to go through an extended dialog. Support undo and redo.
4. **Consistency and standards** Users should not have to wonder whether words, situations or actions mean the same thing in different contexts. Follow platform conventions and accepted standards.
5. **Error prevention** Make it difficult to make errors. Even better than good error messages is a careful design that prevents a problem from occurring in the first place.
6. **Recognition rather than recall** Make objects, actions and options visible. The user should not have to remember information from one part of the dialog to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.
7. **Flexibility and efficiency of use** Allow users to tailor frequent actions. Accelerators – unseen by the novice user – may often speed up the interaction for the expert user to such an extent that the system can cater to both inexperienced and experienced users.

8. **Aesthetic and minimalist design** Dialogs should not contain information that is irrelevant or rarely needed. Every extra unit of information in a dialog competes with the relevant units of information and diminishes their relative visibility.
9. **Help users recognize, diagnose and recover from errors** Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.
10. **Help and documentation** Few systems can be used with no instructions so it may be necessary to provide help and documentation. Any such information should be easy to search, focussed on the user's task, list concrete steps to be carried out, and not be too large.

Once each evaluator has completed their separate assessment, all of the problems are collected and the mean severity ratings calculated. The design team will then determine the ones that are the most important and will receive attention first.

9.3.3 Model-based evaluation

A third expert-based approach is the use of models. Certain cognitive and design models provide a means of combining design specification and evaluation into the same framework. These are discussed in detail in Chapter 12. For example, the GOMS (goals, operators, methods and selection) model predicts user performance with a particular interface and can be used to filter particular design options. Similarly, lower-level modeling techniques such as the keystroke-level model provide predictions of the time users will take to perform low-level physical tasks.

Design methodologies, such as design rationale (see Chapter 6), also have a role to play in evaluation at the design stage. Design rationale provides a framework in which design options can be evaluated. By examining the criteria that are associated with each option in the design, and the evidence that is provided to support these criteria, informed judgments can be made in the design.

Dialog models can also be used to evaluate dialog sequences for problems, such as unreachable states, circular dialogs and complexity. Models such as state transition networks are useful for evaluating dialog designs prior to implementation. These are discussed in detail in Chapter 16.

9.3.4 Using previous studies in evaluation

Experimental psychology and human–computer interaction between them possess a wealth of experimental results and empirical evidence. Some of this is specific to a particular domain, but much deals with more generic issues and applies in a variety of situations. Examples of such issues are the usability of different menu types, the recall of command names, and the choice of icons.

A final approach to expert evaluation exploits this inheritance, using previous results as evidence to support (or refute) aspects of the design. It is expensive to repeat experiments continually and an expert review of relevant literature can avoid the need to do so. It should be noted that experimental results cannot be expected to hold arbitrarily across contexts. The reviewer must therefore select evidence carefully, noting the experimental design chosen, the population of participants used, the analyses performed and the assumptions made. For example, an experiment testing the usability of a particular style of help system using novice participants may not provide accurate evaluation of a help system designed for expert users. The review should therefore take account of both the similarities and the differences between the experimental context and the design under consideration. This is why this is an *expert* review: expertise in the area is required to ensure that correct assumptions are made.

9.4 EVALUATION THROUGH USER PARTICIPATION

The techniques we have considered so far concentrate on evaluating a design or system through analysis by the designer, or an expert evaluator, rather than testing with actual users. However, useful as these techniques are for filtering and refining the design, they are not a replacement for actual usability testing with the people for whom the system is intended: the users. In this section we will look at a number of different approaches to evaluation through user participation. These include empirical or experimental methods, observational methods, query techniques, and methods that use physiological monitoring, such as eye tracking and measures of heart rate and skin conductance.

User participation in evaluation tends to occur in the later stages of development when there is at least a working prototype of the system in place. This may range from a simulation of the system's interactive capabilities, without its underlying functionality (for example, the *Wizard of Oz* technique, which is discussed in Chapter 6, through a basic functional prototype to a fully implemented system. However, some of the methods discussed can also contribute to the earlier design stages, such as requirements capture, where observation and surveying users are important (see Chapter 13).

9.4.1 Styles of evaluation

Before we consider some of the techniques that are available for evaluation with users, we will distinguish between two distinct evaluation styles: those performed under laboratory conditions and those conducted in the work environment or 'in the field'.

Laboratory studies

In the first type of evaluation studies, users are taken out of their normal work environment to take part in controlled tests, often in a specialist usability laboratory (although the 'lab' may simply be a quiet room). This approach has a number of benefits and disadvantages.

A well-equipped usability laboratory may contain sophisticated audio/visual recording and analysis facilities, two-way mirrors, instrumented computers and the like, which cannot be replicated in the work environment. In addition, the participant operates in an interruption-free environment. However, the lack of context – for example, filing cabinets, wall calendars, books or interruptions – and the unnatural situation may mean that one accurately records a situation that never arises in the real world. It is especially difficult to observe several people cooperating on a task in a laboratory situation, as interpersonal communication is so heavily dependent on context (see Section 9.4.2).

There are, however, some situations where laboratory observation is the only option, for example, if the system is to be located in a dangerous or remote location, such as a space station. Also some very constrained single-user tasks may be adequately performed in a laboratory. Finally, and perhaps most commonly, we may deliberately want to manipulate the context in order to uncover problems or observe less used procedures, or we may want to compare alternative designs within a controlled context. For these types of evaluation, laboratory studies are appropriate.

Field studies

The second type of evaluation takes the designer or evaluator out into the user's work environment in order to observe the system in action. Again this approach has its pros and cons.

High levels of ambient noise, greater levels of movement and constant interruptions, such as phone calls, all make field observation difficult. However, the very 'open' nature of the situation means that you will observe interactions between systems and between individuals that would have been missed in a laboratory study. The context is retained and you are seeing the user in his 'natural environment'. In addition, some activities, such as those taking days or months, are impossible to study in the laboratory (though difficult even in the field).

On balance, field observation is to be preferred to laboratory studies as it allows us to study the interaction as it occurs in actual use. Even interruptions are important as these will expose behaviors such as saving and restoring state during a task. However, we should remember that even in field observations the participants are likely to be influenced by the presence of the analyst and/or recording equipment, so we always operate at a slight remove from the natural situation, a sort of Heisenberg uncertainty principle.

This is, of course, a generalization: there are circumstances, as we have noted, in which laboratory testing is necessary and desirable. In particular, controlled experiments can be useful for evaluation of specific interface features, and must normally be conducted under laboratory conditions. From an economic angle, we need to weigh the costs of establishing recording equipment in the field, and possibly disrupting the actual work situation, with the costs of taking one or more participants away from their jobs into the laboratory. This balance is not at all obvious and any study must weigh the loss of contextual information against the increased costs and difficulty of field studies.

9.4.2 Empirical methods: experimental evaluation

One of the most powerful methods of evaluating a design or an aspect of a design is to use a controlled experiment. This provides empirical evidence to support a particular claim or hypothesis. It can be used to study a wide range of different issues at different levels of detail.

Any experiment has the same basic form. The evaluator chooses a hypothesis to test, which can be determined by measuring some attribute of participant behavior. A number of experimental conditions are considered which differ only in the values of certain controlled variables. Any changes in the behavioral measures are attributed to the different conditions. Within this basic form there are a number of factors that are important to the overall reliability of the experiment, which must be considered carefully in experimental design. These include the participants chosen, the variables tested and manipulated, and the hypothesis tested.

Participants

The choice of participants is vital to the success of any experiment. In evaluation experiments, participants should be chosen to match the expected user population as closely as possible. Ideally, this will involve experimental testing with the actual users but this is not always possible. If participants are not actual users, they should be chosen to be of a similar age and level of education as the intended user group. Their experience with computers in general, and with systems related to that being tested, should be similar, as should their experience or knowledge of the task domain. It is no good testing an interface designed to be used by the general public on a participant set made up of computer science undergraduates: they are simply not representative of the intended user population.

A second issue relating to the participant set is the sample size chosen. Often this is something that is determined by pragmatic considerations: the availability of participants is limited or resources are scarce. However, the sample size must be large enough to be considered to be representative of the population, taking into account the design of the experiment and the statistical methods chosen.

Nielsen and Landauer [264] suggest that usability testing with a single participant will find about a third of the usability problems, and that there is little to be gained from testing with more than five. While this may be true of observational studies where the aim is simply to uncover usability issues, it is not possible to discover much about the extent of usability problems from such small numbers. Certainly, if the intention is to run a controlled experiment and perform statistical analysis on the results, at least twice this number is recommended.

Variables

Experiments manipulate and measure variables under controlled conditions, in order to test the hypothesis. There are two main types of variable: those that are ‘manipulated’ or changed (known as the independent variables) and those that are measured (the dependent variables).

Independent variables are those elements of the experiment that are manipulated to produce different conditions for comparison. Examples of independent variables in evaluation experiments are interface style, level of help, number of menu items and icon design. Each of these variables can be given a number of different values; each value that is used in an experiment is known as a *level* of the variable. So, for example, an experiment that wants to test whether search speed improves as the number of menu items decreases may consider menus with five, seven, and ten items. Here the independent variable, number of menu items, has three levels.

More complex experiments may have more than one independent variable. For example, in the above experiment, we may suspect that the speed of the user’s response depends not only on the number of menu items but also on the choice of commands used on the menu. In this case there are two independent variables. If there were two sets of command names (that is, two levels), we would require six experimental conditions to investigate all the possibilities (three levels of menu size × two levels of command names).

Dependent variables, on the other hand, are the variables that can be measured in the experiment, their value is ‘dependent’ on the changes made to the independent variable. In the example given above, this would be the speed of menu selection. The dependent variable must be measurable in some way, it must be affected by the independent variable, and, as far as possible, unaffected by other factors. Common choices of dependent variable in evaluation experiments are the time taken to complete a task, the number of errors made, user preference and the quality of the user’s performance. Obviously, some of these are easier to measure objectively than others. However, the more subjective measures can be applied against predetermined scales, and can be very important factors to consider.

Hypotheses

A hypothesis is a prediction of the outcome of an experiment. It is framed in terms of the independent and dependent variables, stating that a variation in the independent variable will cause a difference in the dependent variable. The aim of the experiment is to show that this prediction is correct. This is done by disproving the null hypothesis, which states that there is no difference in the dependent variable between the levels of the independent variable. The statistical measures described below produce values that can be compared with various levels of significance. If a result is significant it shows, at the given level of certainty, that the differences measured would not have occurred by chance (that is, that the null hypothesis is incorrect).

Experimental design

In order to produce reliable and generalizable results, an experiment must be carefully designed. We have already looked at a number of the factors that the experimenter must consider in the design, namely the participants, the independent and dependent variables, and the hypothesis. The first phase in experimental design then is to choose the hypothesis: to decide exactly what it is you are trying to demonstrate. In doing this you are likely to clarify the independent and dependent variables, in that you will have identified what you are going to manipulate and what change you expect. If your hypothesis does not clearly identify these variables then you need to rethink it. At this stage you should also consider your participants: how many are available and are they representative of the user group?

The next step is to decide on the *experimental method* that you will use. There are two main methods: *between-subjects* and *within-subjects*. In a between-subjects (or *randomized*) design, each participant is assigned to a different condition. There are at least two conditions: the experimental condition (in which the variable has been manipulated) and the control, which is identical to the experimental condition except for this manipulation. This control serves to ensure that it is the manipulation that is responsible for any differences that are measured. There may, of course, be more than two groups, depending on the number of independent variables and the number of levels that each variable can take.

The advantage of a between-subjects design is that any learning effect resulting from the user performing in one condition and then the other is controlled: each user performs under only one condition. The disadvantages are that a greater number of participants are required, and that significant variation between the groups can negate any results. Also, individual differences between users can bias the results. These problems can be handled by a careful selection of participants, ensuring that all are representative of the population and by matching participants between groups.

The second experimental design is within-subjects (or *repeated measures*). Here each user performs under each different condition. This design can suffer from transfer of learning effects, but this can be lessened if the order in which the conditions are tackled is varied between users, for example, group A do first condition followed by second and group B do second condition followed by first. Within-subjects is less costly than between-subjects, since fewer users are required, and it can be particularly effective where learning is involved. There is also less chance of effects from variation between participants.

The choice of experimental method will depend on the resources available, how far learning transfer is likely or can be controlled, and how representative the participant group is considered to be. A popular compromise, in cases where there is more than one independent variable, is to devise a mixed design where one variable is placed between-groups and one within-groups. So, returning to our example of the menu design, the participants would be split into two groups, one for each command set, but each group would perform in three conditions, corresponding to the three possible levels of the number of menu items.

Once we have determined the hypothesis we are trying to test, the variables we are studying, the participants at our disposal, and the design that is most appropriate, we have to decide how we are going to analyze the results we record. There are a number of statistical tests available, and the choice of test is vital to the success of the experiment. Different tests make different assumptions about the data and if an inappropriate test is chosen, the results can be invalid. The next subsection discusses the factors to consider in choosing a statistical test and surveys the most common statistical measures available.

Statistical measures

The first two rules of statistical analysis are to *look* at the data and to *save* the data. It is easy to carry out statistical tests blindly when a glance at a graph, histogram or table of results would be more instructive. In particular, looking at the data can expose *outliers*, single data items that are very different from the rest. Outliers are often the result of a transcription error or a freak event not connected to the experiment. For example, we notice that one participant took three times as long as everyone else to do a task. We investigate and discover that the participant had been suffering from flu on the day of the experiment. Clearly, if the participant's data were included it would bias the results.

Saving the data is important, as we may later want to try a different analysis method. It is all too common for an experimenter to take some averages or otherwise tabulate results, and then throw away the original data. At worst, the remaining statistics can be useless for statistical purposes, and, at best, we have lost the ability to trace back odd results to the original data, as, for example, we want to do for outliers.

Our choice of statistical analysis depends on the type of data and the questions we want to answer. It is worth having important results checked by an experienced statistician, but in many situations standard tests can be used.

Variables can be classified as either *discrete variables* or *continuous variables*. A discrete variable can only take a finite number of values or *levels*, for example, a screen color that can be red, green or blue. A continuous variable can take any value (although it may have an upper or lower limit), for example a person's height or the time taken to complete a task. A special case of continuous data is when they are *positive*, for example a response time cannot be negative. A continuous variable can be rendered discrete by clumping it into classes, for example we could divide heights into short (<5 ft (1.5 m)), medium (5–6 ft (1.5–1.8 m)) and tall (>6 ft (1.8 m)). In many interface experiments we will be testing one design against another. In these cases the independent variable is usually discrete.

The dependent variable is the measured one and subject to random experimental variation. In the case when this variable is continuous, the random variation may take a special form. If the form of the data follows a known *distribution* then special and more powerful statistical tests can be used. Such tests are called *parametric tests* and the most common of these are used when the variation follows the *normal distribution*. This means that if we plot a histogram of the random errors, they will

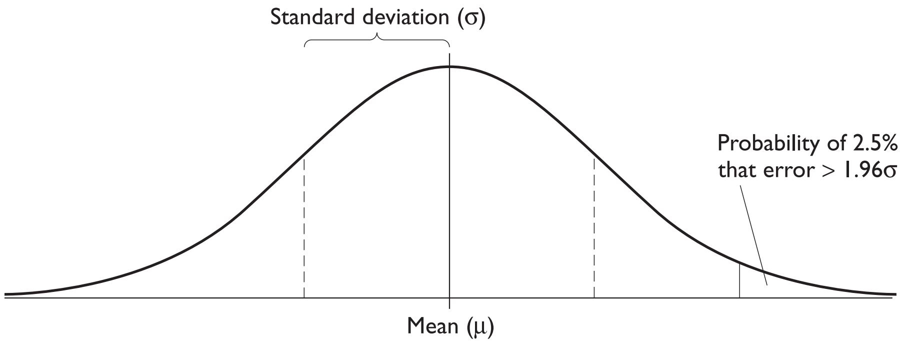


Figure 9.2 Histogram of normally distributed errors

form the well-known bell-shaped graph (Figure 9.2). Happily, many of these tests are fairly *robust*, that is they give reasonable results even when the data are not precisely normal. This means that you need not worry too much about checking normality during early analysis.

There are ways of checking whether data are really normal, but for these the reader should consult a statistics book, or a professional statistician. However, as a general rule, if data can be seen as the sum or average of many small *independent* effects they are likely to be normal. For example, the time taken to complete a *complex* task is the sum of the times of all the minor tasks of which it is composed. On the other hand, a subjective rating of the usability of an interface will not be normal. Occasionally data can be *transformed* to become approximately normal. The most common is the log-transformation, which is used for positive data with near-zero values. As a log-transformation has little effect when the data are clustered well away from zero, many experimenters habitually log-transform. However, this practice makes the results more difficult to interpret and is not recommended.

When we cannot assume that data are normally distributed, we must often resort to *non-parametric* tests. These are statistical tests that make no assumptions about the particular distribution and are usually based purely on the ranking of the data. That is, each item of a data set (for example, 57, 32, 61, 49) is reduced to its rank (3, 1, 4, 2), before analysis begins. Because non-parametric tests make fewer assumptions about the data than parametric tests, and are more resistant to outliers, there is less danger of getting spurious results. However, they are less *powerful* than the corresponding parametric tests. This means that, given the same set of data, a parametric test might detect a difference that the non-parametric test would miss.

A third sort of test is the contingency table, where we classify data by several discrete attributes and then count the number of data items with each attribute combination.

Table 9.1 lists some of the standard tests categorized by the form of independent and dependent variables (discrete/continuous/normal). Normality is not an issue

Table 9.1 Choosing a statistical technique

Independent variable	Dependent variable	
Parametric		
Two valued	Normal	Student's *t* test on difference of means
Discrete	Normal	ANOVA (ANalysis Of VAriance)
Continuous	Normal	Linear (or non-linear) regression factor analysis
Non-parametric		
Two valued	Continuous	Wilcoxon (or Mann–Whitney) rank-sum test
Discrete	Continuous	Rank-sum versions of ANOVA
Continuous	Continuous	Spearman's rank correlation
Contingency tests		
Two valued	Discrete	No special test, see next entry
Discrete	Discrete	Contingency table and chi-squared test
Continuous	Discrete	(Rare) Group independent variable and then as above

for the independent variable, but a special case is when it is discrete with only two values, for example comparing two systems. We cannot describe all the techniques here; for this you should use a standard statistics text, such as one of those recommended in the reading list. The table is only intended to guide you in your choice of test.

An extensive and accurate analysis is no use if it answers the wrong question. Examples of questions one might ask about the data are as follows:

Is there a difference? For example, is one system better than another? Techniques that address this are called *hypothesis testing*. The answers to this question are not simply yes/no, but of the form: 'we are 99% certain that selection from menus of five items is faster than that from menus of seven items'.

How big is the difference? For example, 'selection from five items is 260 ms faster than from seven items'. This is called *point estimation*, often obtained by averages.

How accurate is the estimate? For example, 'selection is faster by 260 ± 30 ms'. Statistical answers to this are in the form of either measures of variation such as the *standard deviation* of the estimate, or *confidence intervals*. Again, the answers one obtains are probabilistic: 'we are 95% certain that the difference in response time is between 230 and 290 ms'.

The experimental design issues we have discussed have been principally addressed at the first question. However, most of the statistical techniques listed above, both parametric and non-parametric, give some answer to one or both of the other questions.

Example of non-parametric statistics

We will not see an example of the use of non-parametric statistics later, so we will go through a small example here. Imagine we had the following data for response times under two conditions:

condition A:	33, 42, 25, 79, 52
condition B:	87, 65, 92, 93, 91, 55

We gather the data together and sort them into order: 25, 33, 42, . . . , 92, 93. We then substitute for each value its rank in the list: 25 becomes 1, 33 becomes 2, etc. The transformed data are then

condition A:	2, 3, 1, 7, 4
condition B:	8, 6, 10, 11, 9, 5

Tests are then carried out on the data. For example, to test whether there is any difference between the two conditions we can use the *Wilcoxon test*. To do this, we take each condition and calculate the sum of ranks, and subtract the least value it could have (that is, $1 + 2 + 3 + 4 + 5 = 15$ for condition A, $1 + 2 + 3 + 4 + 5 + 6 = 21$ for condition B), giving the statistic U:

	rank sum		least		U
condition A:	$(2 + 3 + 1 + 7 + 4)$	−	15	=	2
condition B:	$(8 + 6 + 10 + 11 + 9 + 5)$	−	21	=	28

In fact, the sum of these two U statistics, $2 + 28 = 30$, is the product of the number of data values in each condition 5×6. This will always happen and so one can always get away with calculating only one of the U. Finally, we then take the smaller of two U values and compare it with a set of *critical values* in a book of statistical tables, to see if it is unusually small. The table is laid out dependent on the number of data values in each condition (five and six). The critical value at the 5% level turns out to be 3. As the smallest statistic is smaller than this, we can *reject the null hypothesis* and conclude that there is likely to be a difference between the conditions. To be precise, it says that there is only a 1 in 20 (5%) chance that the data happened by chance. In fact the test is right – the authors constructed random data in the range 1–100 and then subtracted 10 from each of the values in condition A.

An example: evaluating icon designs

Imagine you are designing a new interface to a document-processing package, which is to use icons for presentation. You are considering two styles of icon design and you wish to know which design will be easier for users to remember. One set of icons uses naturalistic images (based on a paper document metaphor), the other uses abstract images (see Figure 9.3). How might you design an experiment to help you decide which style to use?

The first thing you need to do is form a hypothesis: what do you consider to be the likely outcome? In this case, you might expect the natural icons to be easier to recall since they are more familiar to users. We can therefore form the following hypothesis:

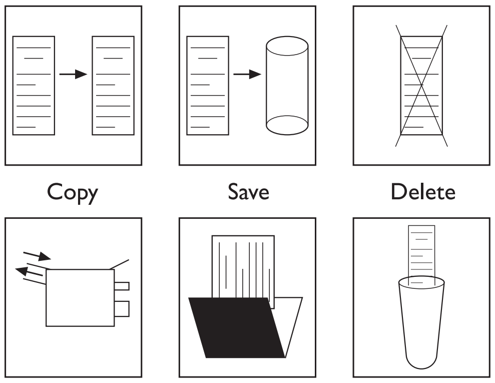


Figure 9.3 Abstract and concrete icons for file operations

Users will remember the natural icons more easily than the abstract ones.

The null hypothesis in this case is that there will be no difference between recall of the icon types.

This hypothesis clearly identifies the independent variable for our experiment: we are varying the style of icon. The independent variable has two levels: natural and abstract. However, when we come to consider the dependent variable, things are not so obvious. We have expressed our hypothesis in terms of users being able to remember *more easily*. How can we measure this? First we need to clarify exactly what we mean by the phrase *more easily*: are we concerned with the user's performance in terms of accurate recall or in terms of speed, for example, or are we looking at more subjective measures like user preference? In this example, we will assume that the speed at which a user can accurately select an icon is an indication of how easily it is remembered. Our dependent variables are therefore the number of mistakes in selection and the time taken to select an icon.

Of course, we need to control the experiment so that any differences we observe are clearly attributable to the independent variable, and so that our measurements of the dependent variables are comparable. To do this, we provide an interface that is identical in every way except for the icon design, and a selection task that can be repeated for each condition. The latter could be either a naturalistic task (such as producing a document) or a more artificial task in which the user has to select the appropriate icon to a given prompt. The second task has the advantage that it is more controlled (there is little variation between users as to how they will perform the task) and it can be varied to avoid transfer of learning. Before performing the selection task, the users will be allowed to learn the icons in controlled conditions: for example, they may be given a fixed amount of time to learn the icon meanings.

The next stage is to decide upon an experimental method. This may depend on the participants that are available, but in this case we will assume that we have sufficient participants from the intended user group. A between-subjects experiment would remove any learning effect for individual participants, but it would be more difficult

Table 9.2 Example experimental results – completion times

Participant number	Presentation order	(1) Natural (s)	(2) Abstract (s)	(3) Participant mean	(4) Natural (1)–(3)	(5) Abstract (2)–(3)
1	AN	656	702	679	−23	23
2	AN	259	339	299	−40	40
3	AN	612	658	635	−23	23
4	AN	609	645	627	−18	18
5	AN	1049	1129	1089	−40	40
6	NA	1135	1179	1157	−22	22
7	NA	542	604	573	−31	31
8	NA	495	551	523	−28	28
9	NA	905	893	899	6	−6
10	NA	715	803	759	−44	44
mean (μ)		698	750	724	−26	26
s.d. (σ)		265	259	262	14	14
			s.e.d. 117		s.e. 4.55	
Student's *t*			0.32 (n.s.)		5.78 (p<1%, two tailed)	

to control for variation in learning style between participants. On balance, therefore, a within-subjects design is preferred, with order of presentation controlled.

So all that remains is to finalize the details of our experiment, given the constraints imposed by these choices. We devise two interfaces composed of blocks of icons, one for each condition. The user is presented with a task (say 'delete a document') and is required to select the appropriate icon. The selection task comprises a set of such presentations. In order to avoid learning effects from icon position, the placing of icons in the block can be randomly varied on each presentation. Each user performs the selection task under each condition. In order to avoid transfer of learning, the users are divided into two groups with each group taking a different starting condition. For each user, we measure the time taken to complete the task and the number of errors made.

Finally, we must analyze our results. Table 9.2 shows a possible set of results for ten participants.[1] The first five had the abstract icons presented first (order AN), and the last five had the natural icons presented first (order NA). Columns (1) and (2) in the table show the completion times for the task using natural and abstract icons respectively. As the times are the result of lots of presentations, we will assume that they are normally distributed. The main independent variable, the icon type, is two valued, suggesting we can use a simple difference of means with Student's *t* test (Table 9.1). In fact, because we have used a *within-subjects* design, there is another independent variable we have to take into account – the participant. This means we

1 Note that these are fabricated results for the purposes of exposition and this is a rather small sample set for real purposes.

have more than one discrete independent variable, and referring again to Table 9.1, we see that this implies we should use *analysis of variance* (*ANOVA*). A full analysis of variance is quite complex, and is ideally done with the aid of a statistics package. However, this experiment is particularly simple, so we can use a simplified analysis.

Look at columns (2) and (3) of Table 9.2. The completion times range from less than 5 minutes (participant 2) to nearly 20 minutes (participant 6), showing a wide variation between individuals. This wide variation emphasizes the importance of the *within*-subjects design. To see how this affects the results, we will first try to analyze them ignoring the fact that each participant performed under each condition. At the end of the table, the mean and standard deviation have been calculated for each condition. These means can then be compared using Student's *t* test. The difference between the means is 52 seconds, but the *standard error of the difference* (s.e.d.) is 117. This is calculated as follows:

$$\text{s.e.d.} = \sqrt{\frac{\sigma_N^2}{n_N} + \frac{\sigma_A^2}{n_A}} = \sqrt{\frac{265^2}{10} + \frac{259^2}{10}} = 117.2$$

where σ_N and σ_A are the standard deviations (s.d.) of the two conditions, and n_N and n_A are the number of data items in each condition (10 in each). The s.e.d. is a measure of the expected variability of the difference between the means, and as we see the actual difference is well within this random variation. Testing the ratio 52/117 against tables of Student's *t* distribution indeed shows that this is not significant.

However, if we glance down the table, we see that in almost every case the time taken with the abstract icons is greater than the time taken for the natural icons. That is, the data seem to support our claim that natural icons are better than abstract ones, but the wide variation between individuals has hidden the effect.

A more sophisticated analysis, a special case of ANOVA, can expose the difference. Looking back at the table, column (3) shows, for each participant, the average of the time they took under the two conditions. This participant mean is then subtracted from the data for each condition, yielding columns (4) and (5). These columns show the effect of the icon design *once the differences between participants have been removed*. The two columns are redundant as they always add up to zero. They show that in all but one case (participant 9) the natural icons are faster than the abstract ones.

Even a non-parametric test would show this as a significant difference at the 5% level, but the use of a *t* test is more precise. We can take either column and see that the column average 26 is much greater than the standard error ($14.4/\sqrt{10}$). The ratio (mean/s.e.) is compared with the Student's *t* table (in statistical tables) using nine degrees of freedom (10 values minus 1 for the mean), and is indeed far greater than the 1% level (3.250); that is, the likelihood of getting our results by chance is less than 1 in 100. So, we reject the null hypothesis that there is no difference and conclude that natural icons are more easily remembered than abstract ones.

In fact, the last statement is not quite correct. What we have shown is that in this experiment natural icons are more *rapidly* remembered. Possibly, if we go on to analyze the errors, these may present a different story. If these error figures were quite large (say 15 errors or more per condition), then we may be able to assume these are normal and use ANOVA. If not, we can either use non-parametric tests, or make use of special tests based on the *binomial distribution*. We will not perform these analyses here. Possibly, looking at the errors we may find that the natural icons have *more* errors – it could well be that they are more rapidly, but less accurately, remembered. It is always worth keeping in mind the difference between the intended purpose of the experiment (to see which is better remembered) and the actual measurements (speed and accuracy).

Finally, one ought to look carefully at the experimental results to see whether there is any other effect that might confuse the results. The graphical presentation of results will help with this, possibly highlighting odd clumps in the data or other irregularities. In this experiment we may want to check to see if there has been any significant *transfer effect* between the first and second condition for each participant. The second set may be faster as the participants are more practiced, or possibly the second set may be slower as learning a second set of icons may be confusing. This will not matter if the effect is uniform – say they always are 15 seconds slower on the second test. But there may be systematic effects. For example, seeing the natural icons first might make it more difficult to learn the abstract ones, but not vice versa. If this were the case, our observed effect may be about the interference between the icon sets, rather than that one is better than the other.

Worked exercise *Design an experiment to test whether adding color coding to an interface will improve accuracy. Identify your hypothesis, participant group, dependent and independent variables, experimental design, task and analysis approach.*

Answer The following is only an example of the type of experiment that might be devised.

Participants Taken from user population.

Hypothesis Color coding will make selection more accurate.

IV (Independent Variable) Color coding.

DV (Dependent Variable) Accuracy measured as number of errors.

Design Between-groups to ensure no transfer of learning (or within-groups with appropriate safeguards if participants are scarce).

Task The interfaces are identical in each of the conditions, except that, in the second, color is added to indicate related menu items. Participants are presented with a screen of menu choices (ordered randomly) and verbally told what they have to select. Selection must be done within a strict time limit when the screen clears. Failure to select the correct item is deemed an error. Each presentation places items in new positions. Participants perform in one of the two conditions.

Analysis *t* test.

Studies of groups of users

So far we have considered the experimental evaluation of single-user systems. Experiments to evaluate elements of group systems bring additional problems. Given the complexities of human–human communication and group working, it is hardly surprising that experimental studies of groups and of groupware are more difficult than the corresponding single-user experiments already considered. For the purpose of discussion, let us assume that we are evaluating a shared application with video connections between the participants and consider some of the problems we will encounter.

The participant groups To organize, say, 10 experiments of a single-user system requires 10 participants. For an experiment involving groups of three, we will, of course, need 30 participants for the same number of experiments. In addition, experiments in group working are often longer than the single-user equivalents as we must allow time for the group to 'settle down' and some rapport to develop. This all means more disruption for participants and possibly more expense payments.

Arranging a mutually convenient slot when both participants and the equipment are available is no mean feat. Often the workstations being used in the experiment will be colleagues' personal systems, so we are trying to accommodate at least six people, not to mention the experimenters themselves.

Not surprisingly, many reports of group working involve only three or four groups. This is obviously a problem for statistical purposes, but not the primary obstacle.

The experimental task Choosing a suitable task is also difficult. We may want to test a variety of different task types: creative, structured, information passing, and so on. Also, the tasks must encourage active cooperation, either because the task requires consensus, or because information and control is distributed among the participants. Obviously, the task also depends on the nature of the groupware system: if it has several available channels, we want to encourage broad use. For example, in the case of shared application with video, it should not be possible (or at least not easy) to perform the task without using the application, otherwise we are simply investigating video conferencing.

Creative tasks such as 'write a short report on . . .' or 'write a research proposal' are often effective, in that the participants must reach agreement, and can be asked to produce their final report using the shared application. Design tasks are also used. For instance, in one experiment, users of the York Conferencer system (see Figure 14.2 in Section 14.4) were asked to redesign a bank layout. A picture of the current layout was used as a background for the spatially arranged electronic pinboard, and the participants made use of this to arrange comments and suggestions close to the features they referred to.

Decision games, as used in management courses, are designed to test and train cooperative activity. They often rely for their success on group coordination, not individual ability. An example of this is the desert survival task, where the participants are told that they have crashed in the desert. They are given a list of items to rank in order of importance for their survival: knife, plastic sheet, etc. The participants must produce *one* list between them, a single knowledgeable participant cannot 'go it alone'. A computerized version of the game of Diplomacy has also been used (see Figure 14.5 in Section 14.4) as it includes aspects of conflict as well as cooperation.

Finally, time-critical simulated process control tasks force a higher pace of interaction as the participants control different parts of the model. An example of this is ARKola [147], a simulated bottling plant, which was used at Xerox PARC to investigate the importance of background noise in complex cooperative control tasks.

Often the chosen task will require extra implementation effort, and in the case of games this may be extensive. This is obviously a strong factor in the choice of a suitable task.

Data gathering Even in a single-user experiment we may well use several video cameras as well as direct logging of the application. In a group setting this is replicated for each participant. So for a three-person group, we are trying to synchronize the recording of six or more video sources and three keystroke logs. To compound matters, these may be spread over different offices, or even different sites. The technical problems are clearly enormous. Four-into-one video recording is possible, storing a different image in each quadrant of the screen, but even this is insufficient for the number of channels we would like.

One way round this is to focus on the participants individually, recording, for each one, the video images that are being relayed as part of the system (assuming there is a video connection) and the sounds that the participant hears. These can then be synchronized with the particular participant's keystrokes and additional video observations. Thus, we can recreate the situation as it appeared *to the participant*. From this recording, we may not be able to interpret the other participants' actions, but at least we have a complete record for one.

Given sufficient recording equipment, this can be repeated for each participant. Happily, the level of synchronization required between participants is not as great as that required for each one individually. One can simply start the recorders' clocks at the same time, but not worry about sub-second accuracy between participants. The important thing is that we can, as it were, relive the experience for each individual.

Analysis In true experimental tradition, we would like to see statistical differences between experimental conditions. We saw earlier that individual differences made this difficult in single-user experiments. If anything, group variation is more extreme. Given randomly mixed groups, one group will act in a democratic fashion; in another, a particular pair will dominate discussion; in a third, one of the participants will act as coordinator, filtering the others' contributions. The level of variation is such that even catastrophic failures under one condition and fabulous successes in another may not always lead to statistically significant results.

As an example of this, imagine we have some quantitative measure of quality of output. We will almost certainly have to use non-parametric tests, so imagine we have found that all the groups under one condition obtained higher scores than any group under the other condition. We would need at least four in each condition to obtain even 5% significance (one tailed). If our results were only slightly less good, say one of the generally better groups performed poorly, we would then require at least five in each condition.

Now this example only considered one condition, and assumed the best possible results. In general, we would expect that the spread between groups within conditions would be greater, and we may want to test more conditions at once. Our 10 groups will have to increase rapidly to stand any chance of statistically significant results. However, we saw above that even gathering 10 experimental groups is a significant problem.

There are three possible solutions to this problem. First, one can use within-group experiments, having each group work under several conditions. We have, of course, the normal problems of such analysis, transfer effects and the like, but we also have more chance of cancelling out the group effect. Secondly, we can look to a micro-analysis of features like gaps between utterances. Such measures are more likely to fit a standard distribution, and thus one can use more powerful parametric tests. In addition, they may be more robust to the large-scale social differences between groups.

The third solution is to opt for a more anecdotal analysis, looking for critical incidents – for example, interesting events or breakdowns – in the data. The concepts and methods for analyzing conversation in Chapter 14 can be used to drive such an analysis. The advantage of this approach is that instead of regarding group differences as a 'problem', they can be included in the analysis. That is, we can begin to look for the systematic ways in which different group structures interact with the communications media and applications they use.

Of course, experiments can be analyzed using both quantitative and qualitative methods. Indeed, any detailed anecdotal analysis of the logs will indicate fruitful measures for statistical analysis. However, if the number of experimental groups is limited, attempts at controlled experiments may not be productive, and may effectively 'waste' the groups used in the control. Given the high costs of group-working experiments, one must choose conditions that are likely to give interesting results, even if statistical analysis proves impossible.

Field studies with groups There are, of course, problems with taking groups of users and putting them in an experimental situation. If the groups are randomly mixed, then we are effectively examining the process of group formation, rather than that of a normal working group. Even where a pre-existent group is used, excluding people from their normal working environment can completely alter their working patterns. For a new system, there may be no 'normal' workplace and all we can do is produce an artificial environment. However, even with a new system we have the choice of producing a 'good' experiment or a naturalistic setting. The traditions of experimental psychology are at odds with those of more qualitative sociological analysis.

It can be argued that group work can only be studied in context. Moving out of the real situation will alter the very nature of the work that is studied. Alternative approaches from the social sciences, such as ethnography, have therefore become popular, particularly in relation to studying group interaction. Ethnography involves very detailed recording of the interactions between people, their environment and each other. The ethnographer attempts to remain outside the situation being studied and does not impose a particular viewpoint on what is observed. This is very different from the experimental perspective with its hypothesis testing. Ethnography is discussed in more detail in Chapter 13.

9.4.3 Observational techniques

A popular way to gather information about actual use of a system is to observe users interacting with it. Usually they are asked to complete a set of predetermined tasks, although, if observation is being carried out in their place of work, they may be observed going about their normal duties. The evaluator watches and records the users' actions (using a variety of techniques – see below). Simple observation is seldom sufficient to determine how well the system meets the users' requirements since it does not always give insight into the their decision processes or attitude. Consequently users are asked to elaborate their actions by 'thinking aloud'. In this section we consider some of the techniques used to evaluate systems by observing user behavior.

Think aloud and cooperative evaluation

Think aloud is a form of observation where the user is asked to talk through what he is doing as he is being observed; for example, describing what he believes is happening, why he takes an action, what he is trying to do.

Think aloud has the advantage of simplicity; it requires little expertise to perform (though can be tricky to analyze fully) and can provide useful insight into problems with an interface. It can also be employed to observe how the system is actually used. It can be used for evaluation throughout the design process, using paper or simulated mock-ups for the earlier stages. However, the information provided is often subjective and may be selective, depending on the tasks provided. The process of observation can alter the way that people perform tasks and so provide a biased view. The very act of describing what you are doing often changes the way you do it – like the joke about the centipede who was asked how he walked . . .

A variation on think aloud is known as *cooperative evaluation* [240] in which the user is encouraged to see himself as a collaborator in the evaluation and not simply as an experimental participant. As well as asking the user to think aloud at the beginning of the session, the evaluator can ask the user questions (typically of the 'why?' or 'what-if?' type) if his behavior is unclear, and the user can ask the evaluator for clarification if a problem arises. This more relaxed view of the think aloud process has a number of advantages:

- the process is less constrained and therefore easier to learn to use by the evaluator
- the user is encouraged to criticize the system
- the evaluator can clarify points of confusion at the time they occur and so maximize the effectiveness of the approach for identifying problem areas.

The usefulness of think aloud, cooperative evaluation and observation in general is largely dependent on the effectiveness of the recording method and subsequent analysis. The record of an evaluation session of this type is known as a *protocol*, and there are a number of methods from which to choose.

Protocol analysis

Methods for recording user actions include the following:

Paper and pencil This is primitive, but cheap, and allows the analyst to note interpretations and extraneous events as they occur. However, it is hard to get detailed information, as it is limited by the analyst's writing speed. Coding schemes for frequent activities, developed during preliminary studies, can improve the rate of recording substantially, but can take some time to develop. A variation of paper and pencil is the use of a notebook computer for direct entry, but then one is limited to the analyst's typing speed, and one loses the flexibility of paper for writing styles, quick diagrams and spatial layout. If this is the only recording facility available then a specific note-taker, separate from the evaluator, is recommended.

Audio recording This is useful if the user is actively 'thinking aloud'. However, it may be difficult to record sufficient information to identify exact actions in later analysis, and it can be difficult to match an audio recording to some other form of protocol (such as a handwritten script).

Video recording This has the advantage that we can see *what* the participant is doing (*as long as* the participant stays within the range of the camera). Choosing suitable camera positions and viewing angles so that you get sufficient detail and yet keep the participant in view is difficult. Alternatively, one has to ask the participant not to move, which may not be appropriate for studying normal behavior! For single-user computer-based tasks, one typically uses two video cameras, one looking at the computer screen and one with a wider focus including the user's face and hands. The former camera may not be necessary if the computer system is being logged.

Computer logging It is relatively easy to get a system automatically to record user actions at a keystroke level, particularly if this facility has been considered early in the design. It can be more difficult with proprietary software where source code is not available (although some software now provides built-in logging and playback facilities). Obviously, computer logging only tells us what the user is doing on the system, but this may be sufficient for some purposes. Keystroke data are also 'semantics free' in that they only tell us about the lowest-level actions, not why they were performed or how they are structured (although slight pauses and gaps can give clues). Direct logging has the advantages that it is cheap (except in terms of disk storage), unobtrusive and can be used for *longitudinal studies*, where we look at one or more users over periods of weeks or months. Technical problems with it are that the sheer volume of data can become unmanageable without automatic analysis, and that one often has to be careful to restore the state of the system (file contents, etc.) before replaying the logs.

User notebooks The participants themselves can be asked to keep logs of activity/ problems. This will obviously be at a very coarse level – at most, records every few minutes and, more likely, hourly or less. It also gives us 'interpreted' records, which have advantages and problems. The technique is especially useful in longitudinal studies, and also where we want a log of unusual or infrequent tasks and problems.

In practice, one uses a mixture of recording methods as they complement one another. For instance, we may keep a paper note of special events and circumstances, even when we have more sophisticated audio/visual recording. Similarly, we may use separate audio recording, even where a video recorder is used, as the quality of specialist audio recording is better than most built-in video microphones. In addition, we may use stereo audio recording, which helps us to locate out-of-screen noises. If one is using a collection of different sources, say audio, video (×2) and keystroke logging, there is considerable difficulty in synchronizing them during play-back. Most video recorders can superimpose an on-screen clock, which can help, but ideally one uses specialized equipment that can automatically synchronize the different sources, possibly merging several video displays onto a single screen. Unfortunately, this sort of equipment is often only available in specialized laboratories.

With both audio and video recording, a major problem is *transcription*. Typing a transcript from a tape is not the same as taped dictation. The conversation will typically consist of part or broken sentences, mumbled words and inarticulated noises. In addition, the transcript will need annotating with the different voices (which may only be clear from context) and with non-verbal items such as pauses, emphases, equipment noises, phones ringing, etc. A good audio-typist will be accustomed to completing mumbled words and correcting ungrammatical sentences – typing *exactly* what is recorded may prove difficult. Some practitioners say that the use of typists is not good practice anyway as the analyst will miss many nuances that are lost in the written transcript. However, if you wish to produce your own typed transcripts from tape, a course in touch-typing is highly recommended.

For video transcription, professional typists are not an option; there is no standard way of annotating video recordings, and the analyst must invent notations to suit the particular circumstances. The scale of this task is not to be underestimated. It is common to talk to practitioners who have tens or hundreds of hours of video recording, but have only analyzed tiny fragments in detail. Of course, the fragments will have been chosen after more extensive perusal of the material, but it certainly removes any idea of comprehensive coverage.

Coding can be introduced to indicate particular events but it is sometimes difficult to determine a suitable coding scheme and to use this consistently, particularly if more than one person is doing the coding. A range of transcribers should therefore test coding schemes to ensure that they are being interpreted appropriately for a particular data set.

Automatic protocol analysis tools

Analyzing protocols, whether video, audio or system logs, is time consuming and tedious by hand. It is made harder if there is more than one stream of data to synchronize. One solution to this problem is to provide automatic analysis tools to support the task. These offer a means of editing and annotating video, audio and system logs and synchronizing these for detailed analysis.

EVA (*Experimental Video Annotator*) is a system that runs on a multimedia workstation with a direct link to a video recorder [220]. The evaluator can devise a set of buttons indicating different events. These may include timestamps and snapshots, as well as notes of expected events and errors. The buttons are used within a recording session by the evaluator to annotate the video with notes. During the session the user works at a workstation and is recorded, using video and perhaps audio and system logging as well. The evaluator uses the multimedia workstation running EVA. On the screen is the live video record and a view of the user's screen (see Figure 9.4). The evaluator can use the buttons to tag interesting events as they occur and can record additional notes using a text editor. After the session, the evaluator can ask to review the tagged segments and can then use these and standard video controls to search the information. Links can be made with other types of record such as audio and system logs. A system such as EVA alleviates the burden of video analysis but it is not without its problems. The act of tagging and annotating events can prevent the evaluator from actually concentrating on the events themselves. This may mean that events are missed or tagged late.

Commercial systems such as Observer Pro from Noldus have similar functionality to EVA; portable versions are now available for use in field studies (www.noldus.com).

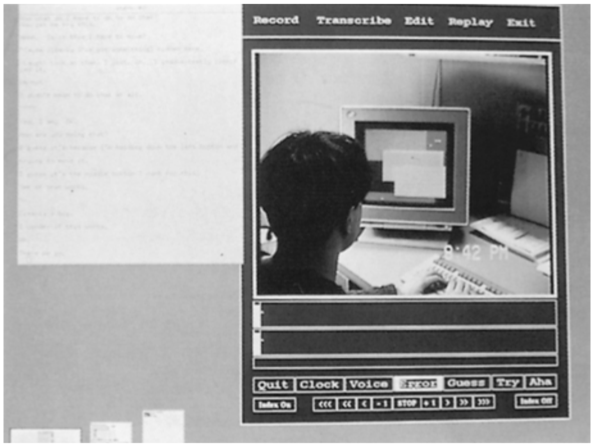


Figure 9.4 EVA: an automatic protocol analysis tool. Source: Wendy Mackay

The *Workplace project* at Xerox PARC [348] also includes a system to aid protocol analysis. The main emphasis here is to support the analysis of synchronized information from different data streams, such as video, audio, notes and diagrams. Each data stream is viewed in an aligned display so that it is possible to compare the records of each for a given point in the interaction. The alignment may be based on timestamps or on an event or action and is implemented using hypertext links.

A third example is DRUM [223], which also provides video annotation and tagging facilities. DRUM is part of the MUSiC (Measuring the Usability of Systems in Context/Metrics for Usability Standards in Computing) toolkit, which supports a complete methodology for evaluation, based upon the application of usability metrics on analytic metrics, cognitive workload, performance and user satisfaction. DRUM is concerned particularly with measuring performance. The methodology provides a range of tools as well as DRUM, including manuals, questionnaires, analysis software and databases.

Systems such as these are extremely important as evaluation tools since they offer a means of handling the data that are collected in observational studies and allowing a more systematic approach to the analysis. The evaluator's task is facilitated and it is likely that more valuable observations will emerge as a result.

Post-task walkthroughs

Often data obtained via direct observation lack interpretation. We have the basic actions that were performed, but little knowledge as to why. Even where the participant has been encouraged to think aloud through the task, the information may be at the wrong level. For example, the participant may say 'and now I'm selecting the undo menu', but not tell us what was wrong to make undo necessary. In addition, a think aloud does not include information such as alternative, but not pursued, actions.

A walkthrough attempts to alleviate these problems, by reflecting the participants' actions back to them after the event. The transcript, whether written or recorded, is replayed to the participant who is invited to comment, or is directly questioned by the analyst. This may be done straightaway, when the participant may actually remember why certain actions were performed, or after an interval, when the answers are more likely to be the participant's post hoc interpretation. (In fact, interpretation is likely even in the former case.) The advantage of a delayed walkthrough is that the analyst has had time to frame suitable questions and focus on specific incidents. The disadvantage is a loss of freshness.

There are some circumstances when the participant cannot be expected to talk during the actual observation, for instance during a critical task, or when the task is too intensive. In these circumstances, the post-task walkthrough is the only way to obtain a subjective viewpoint on the user's behavior. There is also an argument that it is preferable to minimize non-task-related talk during direct observation in order to get as natural a performance as possible. Again this makes the walkthrough essential.

9.4.4 Query techniques

Another set of evaluation techniques relies on asking the user about the interface directly. Query techniques can be useful in eliciting detail of the user's view of a system. They embody the philosophy that states that the best way to find out how a system meets user requirements is to 'ask the user'. They can be used in evaluation and more widely to collect information about user requirements and tasks. The advantage of such methods is that they get the user's viewpoint directly and may reveal issues that have not been considered by the designer. In addition, they are relatively simple and cheap to administer. However, the information gained is necessarily subjective, and may be a 'rationalized' account of events rather than a wholly accurate one. Also, it may be difficult to get accurate feedback about alternative designs if the user has not experienced them, which limits the scope of the information that can be gleaned. However, the methods provide useful supplementary material to other methods. There are two main types of query technique: interviews and questionnaires.

Interviews

Interviewing users about their experience with an interactive system provides a direct and structured way of gathering information. Interviews have the advantages that the level of questioning can be varied to suit the context and that the evaluator can probe the user more deeply on interesting issues as they arise. An interview will usually follow a top-down approach, starting with a general question about a task and progressing to more leading questions (often of the form 'why?' or 'what if?') to elaborate aspects of the user's response.

Interviews can be effective for high-level evaluation, particularly in eliciting information about user preferences, impressions and attitudes. They may also reveal problems that have not been anticipated by the designer or that have not occurred under observation. When used in conjunction with observation they are a useful means of clarifying an event (compare the post-task walkthrough).

In order to be as effective as possible, the interview should be planned in advance, with a set of central questions prepared. Each interview is then structured around these questions. This helps to focus the purpose of the interview, which may, for instance, be to probe a particular aspect of the interaction. It also helps to ensure a base of consistency between the interviews of different users. That said, the evaluator may, of course, choose to adapt the interview form to each user in order to get the most benefit: the interview is not intended to be a controlled experimental technique.

Questionnaires

An alternative method of querying the user is to administer a questionnaire. This is clearly less flexible than the interview technique, since questions are fixed in advance, and it is likely that the questions will be less probing. However, it can be used to reach a wider participant group, it takes less time to administer, and it can be analyzed more rigorously. It can also be administered at various points in the design process, including during requirements capture, task analysis and evaluation, in order to get information on the user's needs, preferences and experience.

Given that the evaluator is not likely to be directly involved in the completion of the questionnaire, it is vital that it is well designed. The first thing that the evaluator must establish is the purpose of the questionnaire: what information is sought? It is also useful to decide at this stage how the questionnaire responses are to be analyzed. For example, do you want specific, measurable feedback on particular interface features, or do you want the user's impression of using the interface?

There are a number of styles of question that can be included in the questionnaire. These include the following:

General These are questions that help to establish the background of the user and his place within the user population. They include questions about age, sex, occupation, place of residence, and so on. They may also include questions on previous experience with computers, which may be phrased as open-ended, multi-choice or scalar questions (see below).

Open-ended These ask the user to provide his own unprompted opinion on a question, for example 'Can you suggest any improvements to the interface?'. They are useful for gathering general subjective information but are difficult to analyze in any rigorous way, or to compare, and can only be viewed as supplementary. They are also most likely to be missed out by time-conscious respondents! However, they may identify errors or make suggestions that have not been considered by the designer. A special case of this type is where the user is asked for factual information, for example how many commands were used.

Scalar These ask the user to judge a specific statement on a numeric scale, usually corresponding to a measure of agreement or disagreement with the statement. For example,

It is easy to recover from mistakes.
Disagree 1 2 3 4 5 Agree

The granularity of the scale varies: a coarse scale (say, from 1 to 3) gives a clear indication of the meaning of the numbers (disagree, neutral and agree). However, it gives no room for varying levels of agreement, and users may therefore be tempted to give neutral responses to statements that they do not feel strongly about but with which they mildly disagree or agree. A very fine scale (say 1 to 10) suffers from the opposite problem: the numbers become difficult to interpret in a consistent way. One user will undoubtedly interpret the scale differently from another. A middle ground is therefore advisable. Scales of 1 to 5 or 1 to 7 have been used effectively. They are fine enough to allow users to differentiate adequately but still retain clarity in meaning. It can help to provide an indication of the meaning of intermediate scalar values. Odd-numbered scales are used most often but it is possible to use even-numbered scales (e.g. 1–6) if the 'neutral' option is not wanted. This does not allow for fence sitting – except decisively by selecting 3½!).

Multi-choice Here the respondent is offered a choice of explicit responses, and may be asked to select only one of these, or as many as apply. For example,

How do you most often get help with the system (tick one)?

Online manual	❑
Contextual help system	❑
Command prompt	❑
Ask a colleague	❑

Which types of software have you used (tick all that apply)?

Word processor	❑
Database	❑
Spreadsheet	❑
Expert system	❑
Online help system	❑
Compiler	❑

These are particularly useful for gathering information on a user's previous experience. A special case of this type is where the offered choices are 'yes' or 'no'.

Ranked These place an ordering on items in a list and are useful to indicate a user's preferences. For example,

Please rank the usefulness of these methods of issuing a command (1 most useful, 2 next, 0 if not used).

Menu selection	❑
Command line	❑
Control key accelerator	❑

These question types are all useful for different purposes, as we have noted. However, in order to reduce the burden of effort on the respondent, and so encourage a high response rate amongst users, it is best to use closed questions, such as scalar, ranked or multi-choice, as much as possible. These provide the user with alternative responses and so reduce the effort required. They also have the advantage of being easier to analyze. Responses can be analyzed in a number of ways, from determining simple percentages for each response, to looking at correlations and factor analysis. For more detail on available methods the reader is referred to the recommended reading list at the end of the chapter.

Whatever type of questionnaire is planned, it is wise to carry out a pilot study. This allows any problems with the questionnaire design to be ironed out before the questionnaire is distributed to potentially hundreds of users! The questionnaire should be tested on four or five users to see if the questions are comprehensible and the results are as expected and can be used in the manner intended. If users seem to be misunderstanding a particular question, it can then be rephrased (and retested) before the final version is sent out.

Distribution of questionnaires can also be problematic. It is important that the respondents are representative of the user population but you also need to ensure that you are able to reach as many potential respondents as possible. Return rate for questionnaires is quite low (often 25–30%) so many more need to be sent out to get a reasonable return. Questionnaires should ideally be distributed to a random subset of the user population. So, for example, if the population is all workers in a company, one may choose to send a questionnaire to every fourth person on an alphabetically ordered personnel list. However, questionnaires are now often distributed via the internet, either by email, where potential respondents can be selected randomly, or via a website, where the respondents are limited to those who visit the site and who may not be representative. In practice, questionnaire respondents are self-selecting anyway, in that only those who choose to respond are included in the study; if the questionnaire is designed to capture demographic information about each respondent then the level of representativeness (or otherwise) can be determined from the responses.

Worked exercise *You have been asked to compare user performance and preferences with two different learning systems, one using hypermedia (see Chapter 21), the other sequential lessons. Design a questionnaire to find out what the users think of the system. How would you go about comparing user performance with these two systems?*

Answer Assume that all users have used both systems.

Questionnaire
Consider the following questions in designing the questionnaire:

- what information is required?
- how is the questionnaire to be analyzed?

You are particularly interested in user preferences so questions should focus on different aspects of the systems and try to measure levels of satisfaction. The use of scales will make responses for each system easier to compare.

Table 9.3 shows an example questionnaire.

To test performance you would design an experiment where two groups of participants learn the same material using the two systems, and test how well they have learned (using a standard measurable test).

Participants User group

IV (Independent Variable) Style of learning system

DV (Dependent Variable) Performance (measured as test score)

Design Between-subjects design

Table 9.3 Questionnaire to compare two systems

PART I: Repeat for each system

Indicate your agreement or disagreement with the following statements. (1 indicates complete disagreement and 5 complete agreement.)

The system tells me what to do at every point.
Disagree 1 2 3 4 5 Agree

It is easy to recover from mistakes.
Disagree 1 2 3 4 5 Agree

It is easy to get help when needed.
Disagree 1 2 3 4 5 Agree

I always know what the system is doing.
Disagree 1 2 3 4 5 Agree

I always know where I am in the training material.
Disagree 1 2 3 4 5 Agree

I have learned the material well using the system.
Disagree 1 2 3 4 5 Agree

I could have learned the material more effectively using a book.
Disagree 1 2 3 4 5 Agree

I always know how well I am doing.
Disagree 1 2 3 4 5 Agree

PART II: Comparing both systems:

Which system (choose 1) was most:

Helpful to use	A	B
Efficient to use	A	B
Enjoyable to use	A	B

Please add any comments you have about either system:

9.4.5 Evaluation through monitoring physiological responses

One of the problems with most evaluation techniques is that we are reliant on observation and the users telling us what they are doing and how they are feeling. What if we were able to measure these things directly? Interest has grown recently in the use of what is sometimes called objective usability testing, ways of monitoring physiological aspects of computer use. Potentially this will allow us not only to see more clearly exactly what users do when they interact with computers, but also to measure how they feel. The two areas receiving the most attention to date are eye tracking and physiological measurement.

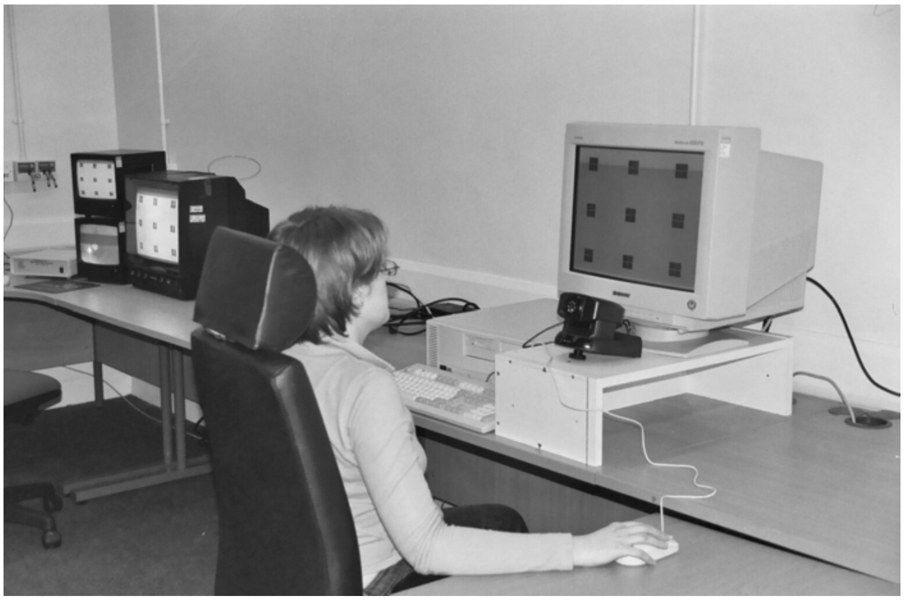


Figure 9.5 Eye-tracking equipment. Source: Courtesy of J. A. Renshaw

Eye tracking for usability evaluation

Eye tracking has been possible for many years, but recent improvements in hardware and software have made it more viable as an approach to measuring usability. The original eye trackers required highly invasive procedures where eye caps were attached to the cornea under anaesthetic. Clearly inappropriate for usability testing! Modern systems vary: some use a head-mounted camera to monitor the eye, but the most sophisticated do not involve any contact between the equipment and the participant, with the camera and light sources mounted in desk units (see Figures 9.5, 9.6) [112].

Furthermore, there have been rapid improvements in the software available both for the control of eye-tracking equipment and the analysis and visualization of the large volumes of data it produces.

Eye movements are believed to reflect the amount of cognitive processing a display requires and, therefore, how easy or difficult it is to process [150]. So measuring not only where people look, but also their patterns of eye movement, may tell us which areas of a screen they are finding easy or difficult to understand. Eye movement measurements are based on fixations, where the eye retains a stable position for a period of time, and saccades, where there is rapid ballistic eye movement from one point of interest to another. There are many possible measurements related to usability evaluation including:

Number of fixations The more fixations the less efficient the search strategy.

Fixation duration Longer fixations may indicate difficulty with a display.

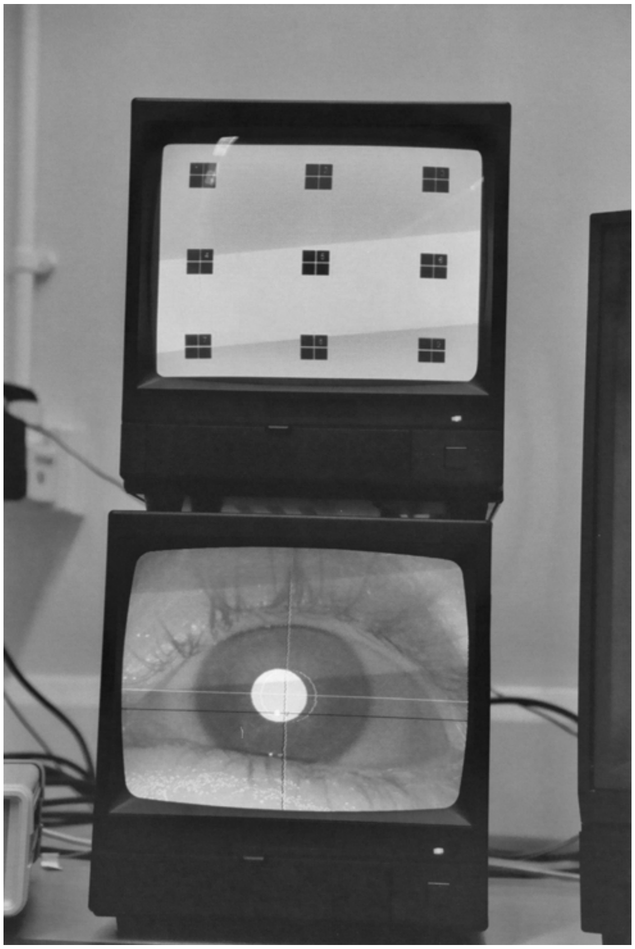


Figure 9.6 Calibrating the eye tracker. Source: Courtesy of J. A. Renshaw

Scan path indicating areas of interest, search strategy and cognitive load. Moving straight to a target with a short fixation at the target is the optimal scan path but plotting scan paths and fixations can indicate what people look at, how often and for how long.

Eye tracking for usability is still very new and equipment is prohibitively expensive for everyday use. However, it is a promising technique for providing insights into what really attracts the eye in website design and where problem areas are in system use. More research is needed to interpret accurately the meaning of the various eye movement measurements, as well as to develop more accessible and robust equipment. But, given the potential for gathering new data measurements relatively unobtrusively, it is likely that eye tracking will become part of the standard equipment for usability laboratories in the coming few years.

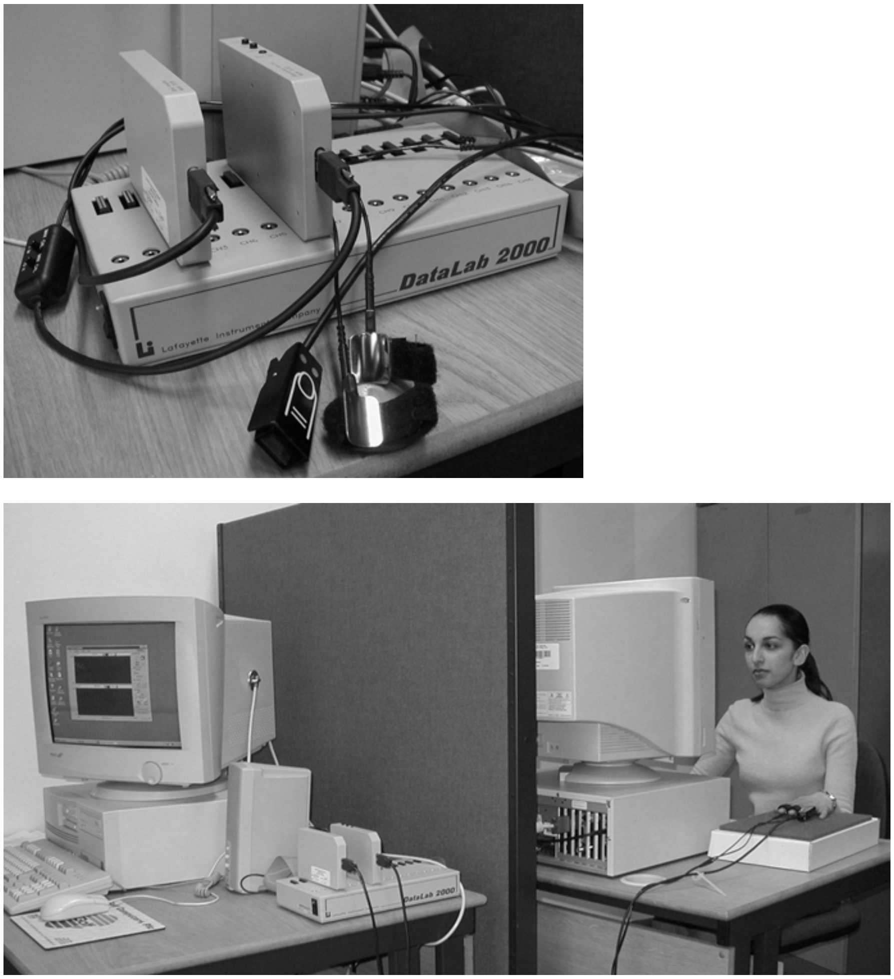


Figure 9.7 Data Lab Psychophysiology equipment showing some of the sensors (above) and a typical experimental arrangement (below) with sensors attached to the participant's fingers and the monitoring software displayed on the evaluator's machine. Source: Courtesy of Dr R. D. Ward

Physiological measurements

As we saw in Chapter 1, emotional response is closely tied to physiological changes. These include changes in heart rate, breathing and skin secretions. Measuring these physiological responses may therefore be useful in determining a user's emotional response to an interface [288, 363]. Could we determine which interaction events really cause a user stress or which promote relaxation?

Physiological measurement involves attaching various probes and sensors to the user (see Figure 9.7). These measure a number of factors: