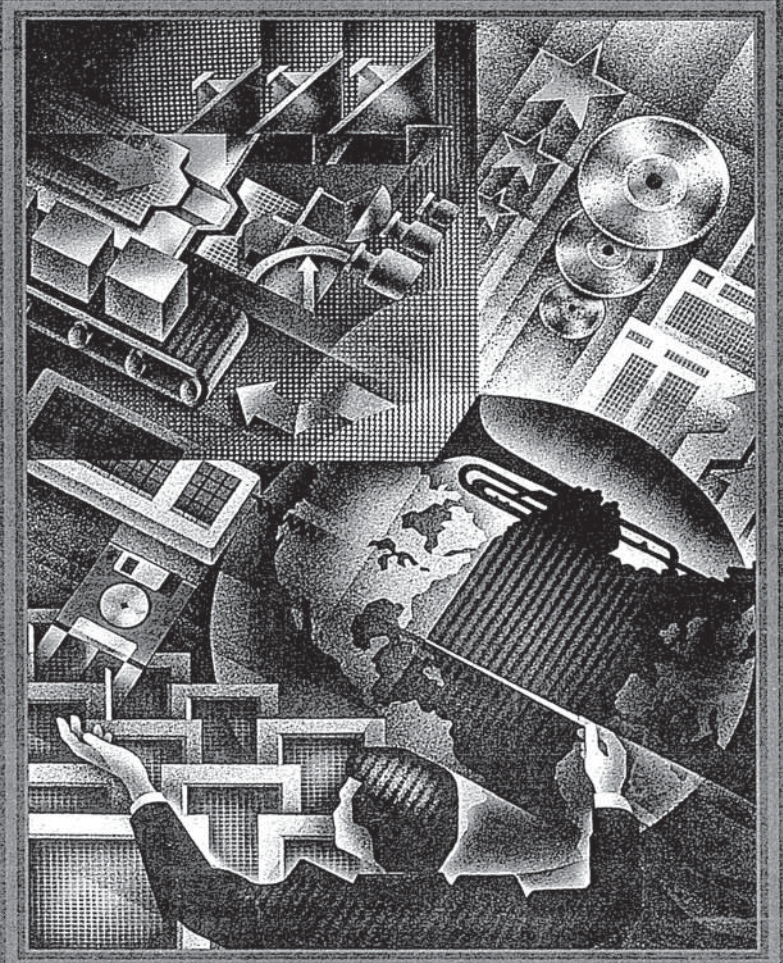
Designing the User Interface

Strategies for Effective Human-Computer Interaction



Third Edition

Ben Shneiderman

Designing the User Interface

Strategies for Effective Human-Computer Interaction

Third Edition

Ben Shneiderman
The University of Maryland

ADDISON-WESLEY

An imprint of Addison Wesley Longman, Inc.

Reading, Massachusetts • Harlow, England • Menlo Park, California
Berkeley, California • Don Mills, Ontario • Sydney • Bonn • Amsterdam
Tokyo • Mexico City

Library of Congress Cataloging-in-Publication Data

Shneiderman, Ben.
Designing the user interface : strategies for effective human
-computer-interaction / Ben Shneiderman. -- 3rd ed.
p. cm.
Includes bibliographical references and index.
ISBN 0-201-69497-2
1. Human-computer interaction. 2. User interfaces (Computer
systems) I. Title.
QA76.9.H85S54 1998
004'.01'9--dc21 96-37974
CIP

Access the latest information about Addison-Wesley titles from our World Wide Web site: http://www.awl.com/cseng

Chapter opener illustrations from art provided by Mark Kostabi. Reproduced with permission.

Cover art © Boris Lyubner/SIS

Reprinted with corrections, March 1998.

6 7 8 9 10-MA-01009998

Preface to the Third Edition

Designing the User Interface is intended primarily for designers, managers, and evaluators of interactive systems. It presents a broad survey of designing, implementing, managing, maintaining, training, and refining the user interface of interactive systems. The book's second audience is researchers in human–computer interaction, specifically those who are interested in human performance with interactive systems. These researchers may have backgrounds in computer science, psychology, information systems, library science, business, education, human factors, ergonomics, or industrial engineering; all share a desire to understand the complex interaction of people with machines. Students in these fields also will benefit from the contents of this book. It is my hope that this book will stimulate the introduction of courses on user-interface design in all these and other disciplines. Finally, serious users of interactive systems will find that the book gives them a more thorough understanding of the design questions for user interfaces. My goals are to encourage greater attention to the user interface and to help develop a rigorous science of user-interface design.

Since publication of the first two editions of this book in 1986 and 1992, researchers in the field of human–computer interaction and practitioners of user-interface design have grown more numerous and influential. The quality of interfaces has improved greatly, and the community of users has grown dramatically. Researchers and designers could claim success, but user expectations are higher and the applications are more demanding. Today's interfaces are good, but novice and expert users still experience anxiety and frustration all too often. To achieve the goal of universal access, designers will have to continue to work harder. This book is meant to help them keep up the momentum, and thus to encourage further progress.

Keeping up with the innovations in human–computer interaction is a demanding task. Requests for an update to my second edition began shortly after its publication, but I had to wait until a sabbatical year allowed me to set aside enough time to complete this third edition. I've gone to the library, the World Wide Web, conferences, and colleagues to harvest information, and then returned to my keyboard to write. My first drafts were only a starting point to generate feedback from colleagues, practitioners, and students. The work was intense and satisfying.

New in the Third Edition

Comments from instructors who used the second edition were influential in my revisions of the structure. Since many courses include design, evaluation,

and construction projects, the chapters on development methodologies, evaluation techniques, and software tools were moved toward the beginning. Since direct manipulation is the dominant user-interface style, it is presented first, followed by menus, form fillin, and command languages. The material on computer-supported cooperative work has changed dramatically as research ideas and prototypes have become commercial tools. Information visualization is still in its early phases, but vigorous research and emerging commercial activity are widespread. The closing chapter on the rapidly growing World Wide Web is totally new.

Instructors wanted more guidelines and summary tables; these elements are now shown in boxes throughout the book. The Practitioner Summaries and Researcher Agendas remain popular; they have been updated. The references have been expanded and freshened with many new sources, with classic papers still included. Because some of the previously cited works were difficult to find, a much larger percentage of the references now are widely available sources. Figures—especially those showing screen designs—age quickly. In this edition, numerous new user interfaces are shown, many in full color.

Readers will see the dynamism of human–computer interaction reflected in the substantial changes to this third edition. Controversy continues about the future of speech input and output, natural-language interaction, anthropomorphic design, and agents. I emphasize empirical reports, try to present both sides fairly, and offer my opinions.

The presence of the World Wide Web has a profound effect on researchers, designers, educators, and students. I want to encourage intense use of the web by all these groups and to ease integration of the web into common practice. However, the volatility of the web is not in harmony with the permanence of printed books. Publishing website URLs in the book would have been risky, because changes are made daily. For these and other reasons, with the cooperation of my publisher and Prof. Blaise Liffick (Millersville University), we have established an ambitious web site (http://www.aw.com/DTUI) to accompany this book. It contains pointers to web sites related to each chapter's topics, updates on fast-changing topics, interesting reviews, and instructional support. Exercises, homework assignments, projects, and examination questions are just a few of the elements of this evolving site. Contributions from professionals, faculty, and students are making this resource increasingly valuable, and the community using it is lively and growing. I hope that every reader will visit the site, will participate in discussion groups, and will contribute to it. Send us your ideas and contributions.

Ways to Use This Book

I hope that practitioners and researchers who read this book will want to keep it on their shelves to consult when they are working on a new topic or seeking pointers to the literature.

Instructors may choose to assign the full text in the order that I present it, or to make selections from it. The opening chapter is a good starting point for most students, but instructors may take different paths depending on their disciplines. For example, instructors might emphasize the following chapters, listed by area:

- Computer science: 2, 5, 6, 13, 14, 15
- Psychology: 2, 4, 9, 10, 14
- Library and information science: 2, 4, 12, 15, 16
- Business and information systems: 3, 4, 14, 15
- Education technology: 2, 4, 11, 12, 14, 16
- Communication arts and media studies: 4, 11, 12, 16
- Technical writing and graphic design: 3, 4, 11, 12, 15, 16

The book's web site provides syllabi from many instructors, and offers supplemental teaching materials.

Acknowledgments

Writing is a lonely process; revising is a social one. I am grateful to the many colleagues and students who contributed their suggestions. My close daily partners at the University of Maryland have the greatest influence and my deepest appreciation: Gary Marchionini, Kent Norman, Catherine Plaisant, and Anne Rose. I give special thanks to Charles Kreitzberg and Jenny Preece for their personal and professional support. Other major contributors of useful comments include Richard Bellaver, Tom Bruns, Stephan Greene, Jesse Heines, Eser Kandogan, Chris North, Arkady Pogostkin, Richard Potter, Marilyn Saltzman, Michael Spring, Egemen Tanin, and Craig Wills. The many people and organizations that provided figures are acknowledged in the relevant captions.

I also appreciate the students around the world who sent me comments and suggestions. Their provocative questions about our growing discipline and profession encourage me daily.

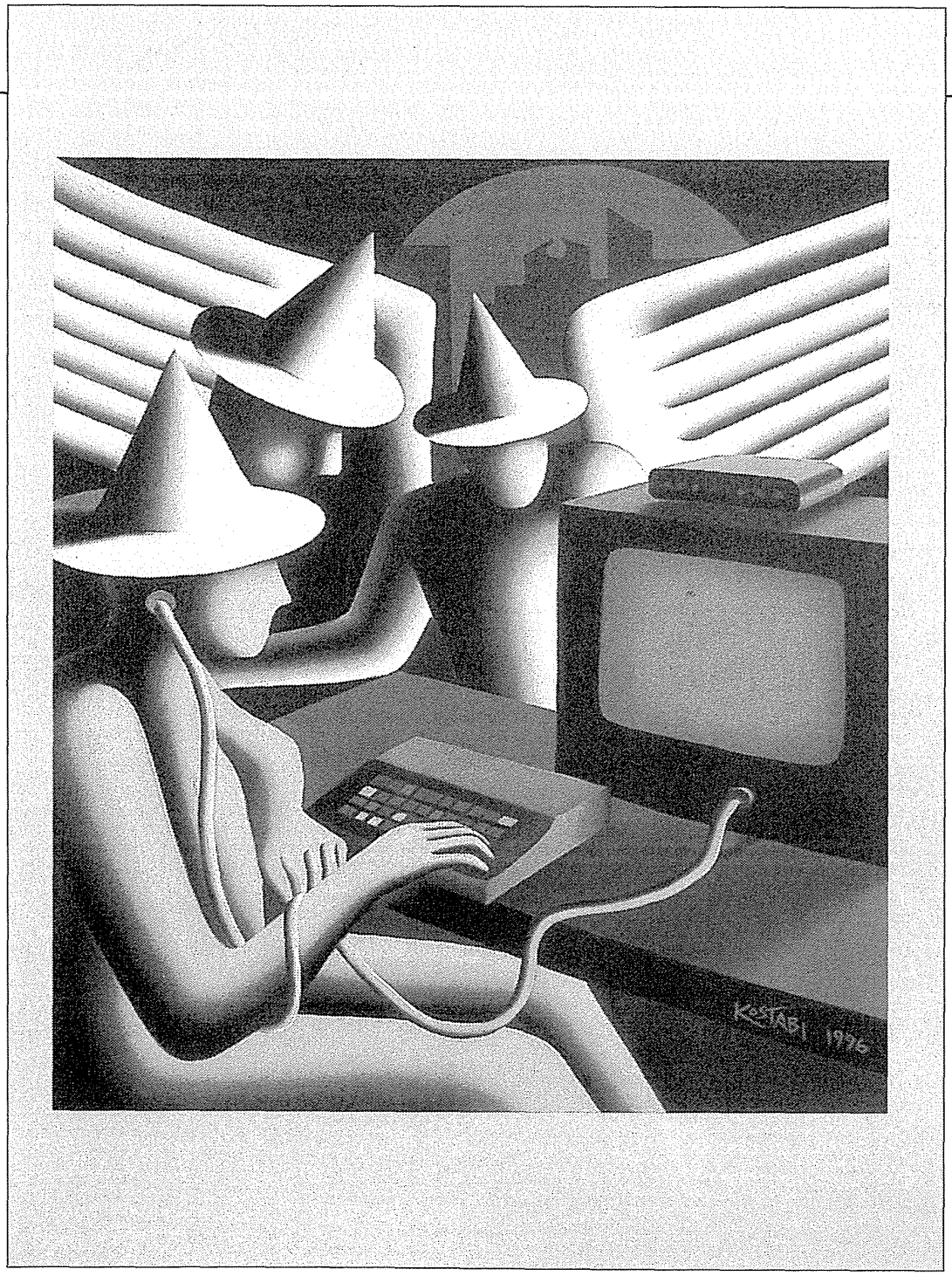
Ben Shneiderman (ben@cs.umd.edu)

Contents

Designing the User Interface

Strategies for Effective Human-Computer Interaction

Third Edition



Mark Kostabi, *Technological Obsession (Moon and Mystery)*, 1996

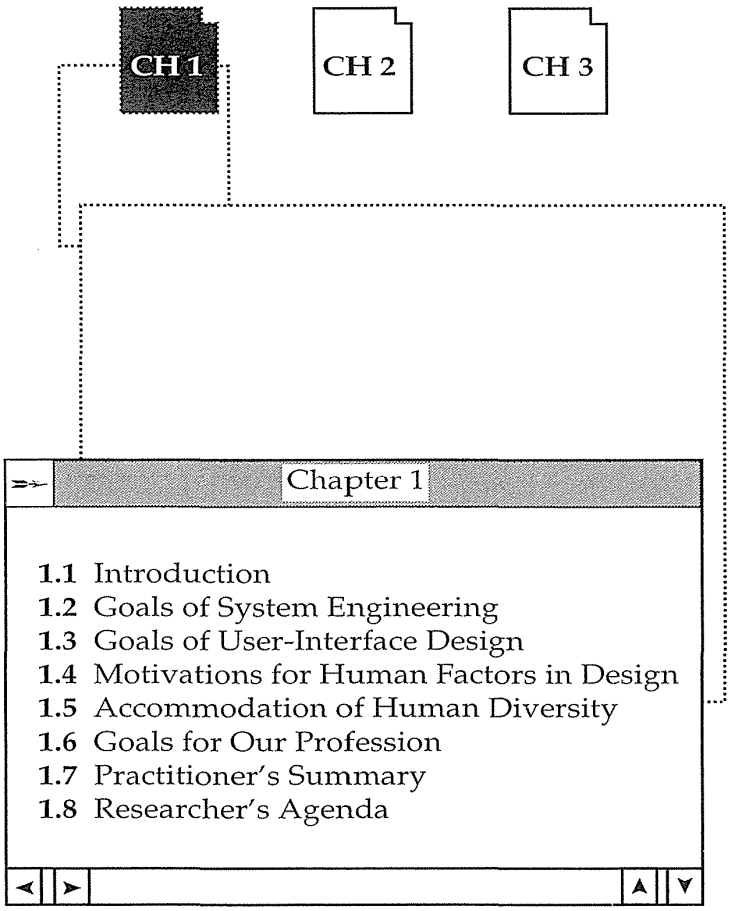
CHAPTER

1

Human Factors of Interactive Software

Designing an object to be simple and clear takes at least twice as long as the usual way. It requires concentration at the outset on how a clear and simple system would work, followed by the steps required to make it come out that way—steps which are often much harder and more complex than the ordinary ones. It also requires relentless pursuit of that simplicity even when obstacles appear which would seem to stand in the way of that simplicity.

T. H. Nelson, ***The Home Computer Revolution,*** **1977**



1.1 Introduction

New technologies provide extraordinary—almost supernatural—powers to those people who master them. Computer systems and accessible interfaces are still new technologies that are being rapidly disseminated. Great excitement spreads as designers provide remarkable functions in carefully crafted interactive and networked systems. The opportunities for youthful system builders and mature entrepreneurs are substantial, and the impacts on individuals and organizations are profound.

Like early photography equipment or automobiles, computers have been available only to people who were willing to devote effort to mastering the technology. Harnessing the computer's power is a task for designers who understand the technology and are sensitive to human capacities and needs.

Human performance in the use of computer and information systems will remain a rapidly expanding research and development topic in the coming decades. This interdisciplinary journey of discovery combines the data-gathering methods and intellectual framework of experimental psychology with the powerful and widely used tools developed from computer science. Contributions also accrue from educational and industrial psychologists, instructional and graphic designers, technical writers, experts in human factors or ergonomics, and adventuresome anthropologists or sociologists.

Applications developers who apply human-factors principles and processes are producing exciting interactive systems. Provocative ideas emerge in the pages of the numerous thick computer magazines, the shelves of the proliferating computer stores, and the menus of the expanding computer networks. User interfaces produce corporate success stories and Wall Street sensations such as Netscape, America Online, or Lycos. They also produce intense competition (with Microsoft as a favorite enemy), copyright-infringement suits (such as Apple's suit against Microsoft covering the Windows interface), mega-mergers (such as Bell Atlantic and NYNEX), takeovers (such as IBM grabbing Lotus), and international liaisons (such as British Telecom's link to MCI).

At an individual level, user interfaces change many people's lives: doctors can make more accurate diagnoses, children can learn more effectively, graphic artists can explore more creative possibilities, and pilots can fly airplanes more safely. Some changes, however, are disruptive; too often, users must cope with frustration, fear, and failure when they encounter excessive complexity, incomprehensible terminology, or chaotic layouts.

The steadily growing interest in user-interface design spans remarkably diverse systems (Figs. 1.1 to 1.7 and Color Plates A1 to A6). Word processors and desktop-publishing tools are used routinely, and many businesses employ photo scanning and image-manipulation software. Electronic mail, computer conferencing, and the World Wide Web have provided new communication media. Digital image libraries are expanding in applications from medicine to space exploration. Scientific visualization and simulator workstations allow safe experimentation and inexpensive training. Electronic spreadsheets and decision-support systems serve as tools for analysts from many disciplines. Educational and public access to information from museum kiosks or government sources is expanding. Commercial systems include inventory, personnel, reservations, air traffic, and electric-utility control. Computer-assisted software-engineering tools and programming environments allow rapid prototyping, as do computer-assisted design, manufacturing, and engineering workstations. Most of us use various consumer electronics, such as VCRs, telephones, cameras, and appliances. Art, music, sports, and entertainment all are assisted or enhanced by computer systems.

Practitioners and researchers in many fields are making vital contributions. Academic and industrial theorists in computer science, psychology, and human factors are developing perceptual, cognitive, and motor theories

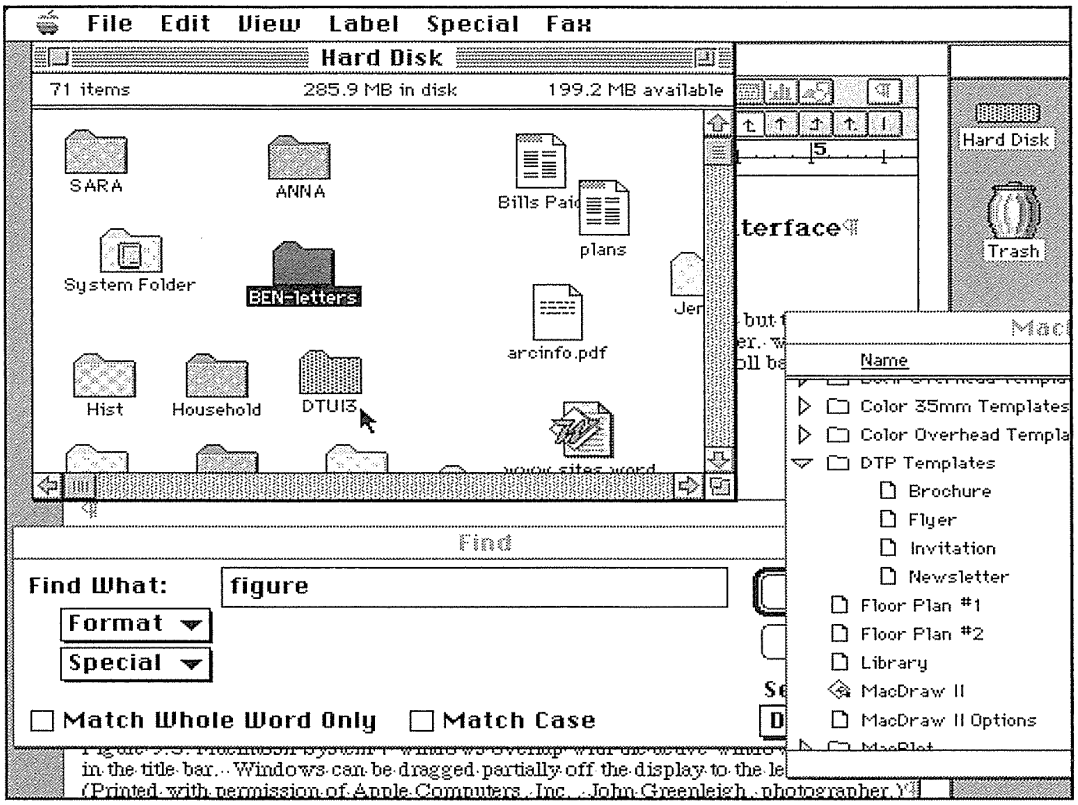


Figure 1.1

Macintosh System 7.5. The active window, which shows stripes in the title bar, is on top. Windows can be dragged partially off the display to the left, right, and bottom. File and folder icons can be dragged to new folders or to the trashcan for deletion. (Used with permission of Apple Computer, Inc., Cupertino, CA.)

and models of human performance, while experimenters are collecting empirical data.

Software designers are exploring how best to organize information graphically. They are developing query languages and visually attractive facilities for input, search, and output. They are using sound (such as music and voice), three-dimensional representations, animation, and video to improve the appeal and information content of interfaces. Techniques such as direct manipulation, telepresence, and virtual realities may change the ways that we interact with and think about computers.

Hardware developers and system builders are offering novel keyboard designs and pointing devices, as well as large, high-resolution color displays. They are designing systems that both provide rapid response times for increasingly complex tasks and have fast display rates and smooth transitions

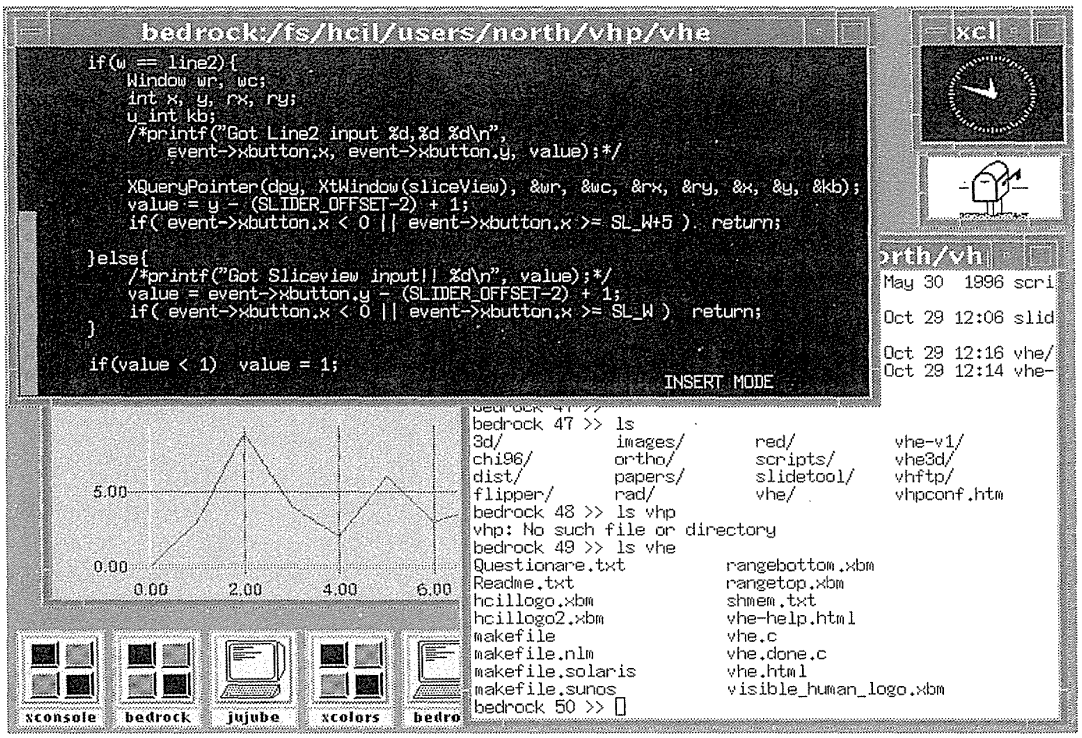


Figure 1.2

Unix Motif environment. A programmer is shown at work.

for increasingly complex 3-dimensional manipulations. Technologies that allow speech input and output, gestural input, and tactile or force-feedback output increase ease of use, as do input devices such as the touchscreen and stylus.

Developers with an orientation toward educational psychology, instructional design, and technical writing are creating engaging online tutorials, training, reference manuals, demonstrations and sales materials, and are exploring novel approaches to group lectures, distance learning, personalized experiential training, and video presentations. Graphic designers are actively engaged in visual layout, color selection, and animation. Sociologists, anthropologists, philosophers, policy makers, and managers are dealing with organizational impact, computer anxiety, job redesign, retraining, distributed teamwork, computer-supported cooperation strategies, work-at-home schemes, and long-term societal changes.

We are living in an exciting time for developers of user interfaces. The hardware and software foundations for the bridges and tunnels have been built. Now the roadway can be laid and the stripes painted to make way for the heavy traffic of eager users.

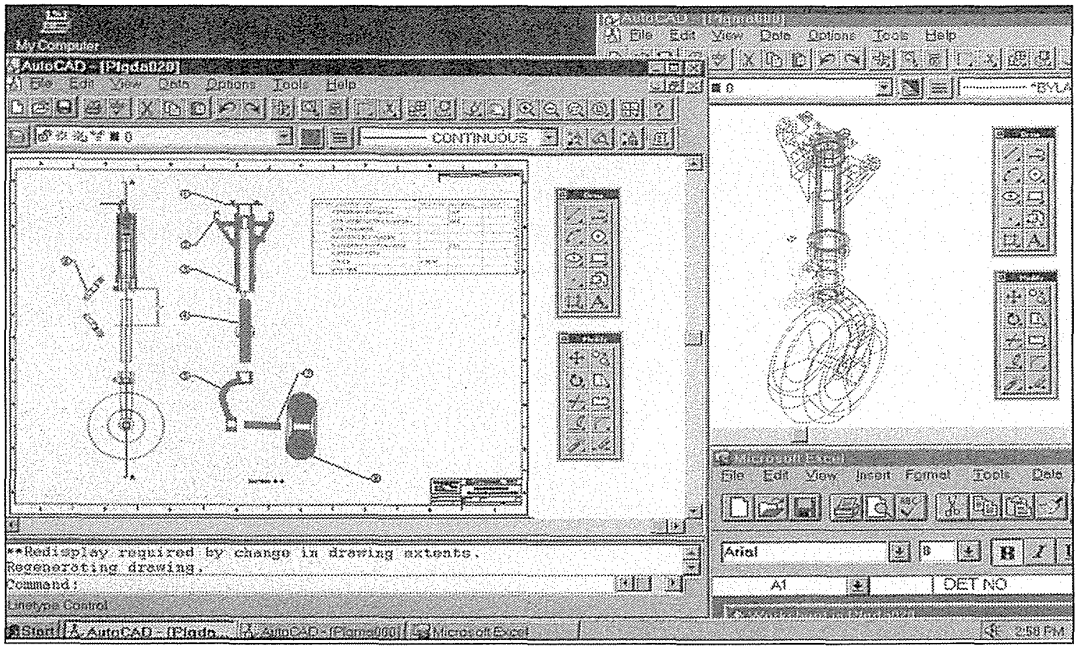


Figure 1.3

AutoCAD R13 for Windows. This design environment has multiple windows and palettes for an aircraft landing-gear assembly. (Used with permission of AutoDesk, San Rafael, CA.)

The rapid growth of interest in user-interface design is international in scope. In the United States, the Association for Computing Machinery (ACM) Special Interest Group in Computer Human Interaction (SIGCHI) had more than 6000 members in 1997. The annual CHI conferences draw almost 2500 people. The Usability Professionals Association focuses on commercial approaches, and the Human Factors & Ergonomics Society, the American Society for Information Science, and other professional groups attend to research on human–computer interaction. Regular conferences in Europe, Japan, and elsewhere draw substantial audiences of researchers and practitioners. In Europe, the ESPRIT project devotes approximately 150 person-years of effort per year to the topic. In Japan, the Ministry of International Trade and Industry promotes commercially-oriented projects and consortia among many companies.

This chapter gives a broad overview of human–computer interaction from practitioner and research perspectives. Specific references cited in the chapter appear on page 33, and a set of general references begins on page 35.

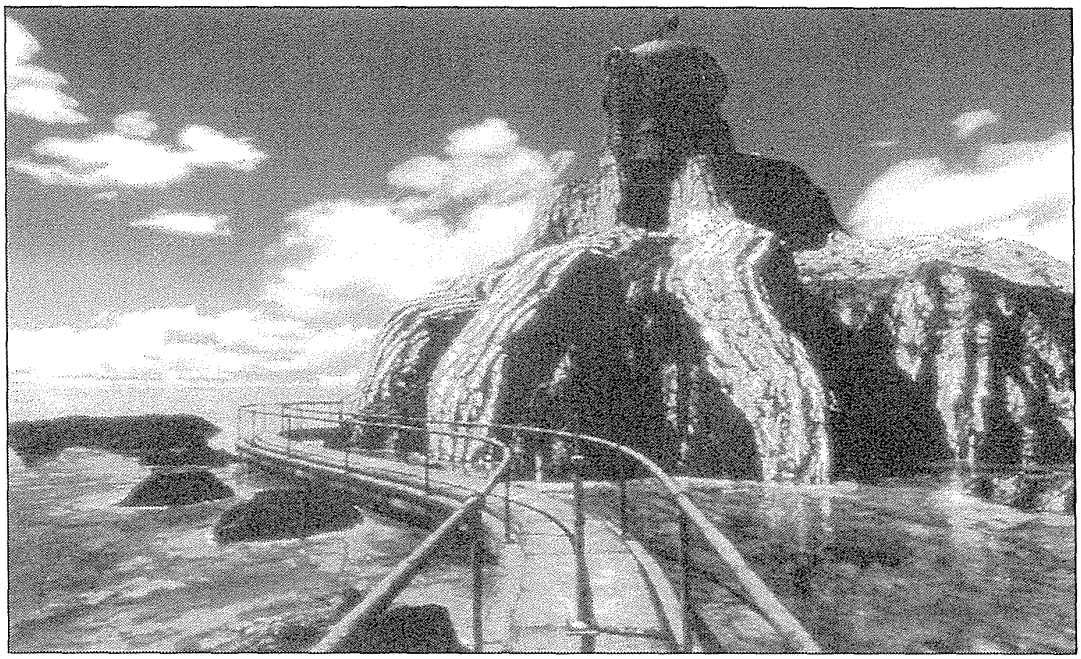


Figure 1.4

Realistic textures add to this outdoor setting that leads the player to one of the islands making up the world of *Riven: The Sequel to MYST* (Copyright Cyan, Inc.) MYST (1994) and Riven (1997), created by Rand and Robyn Miller, are entrancing environments that bridge literary styles with video games. (Used with permission of Broderbund, Inc.)

1.2 Goals of System Engineering

The high-level goal of making the user's quality of life better (see Afterword) is important to keep in mind, but designers have more specific goals. Every designer wants to build a high-quality interactive system that is admired by colleagues, celebrated by users, circulated widely, and imitated frequently. Appreciation comes not from flamboyant promises or stylish advertising, but rather from inherent quality features that are achieved through thoughtful planning, sensitivity to user needs, and diligent testing.

Managers can promote attention to user-interface issues by selection of personnel, preparation of schedules and milestones, construction and application of guidelines documents, and commitment to testing. Designers then propose multiple design alternatives for consideration, and the leading contenders are subjected to further development and testing (see Chapters 3 and

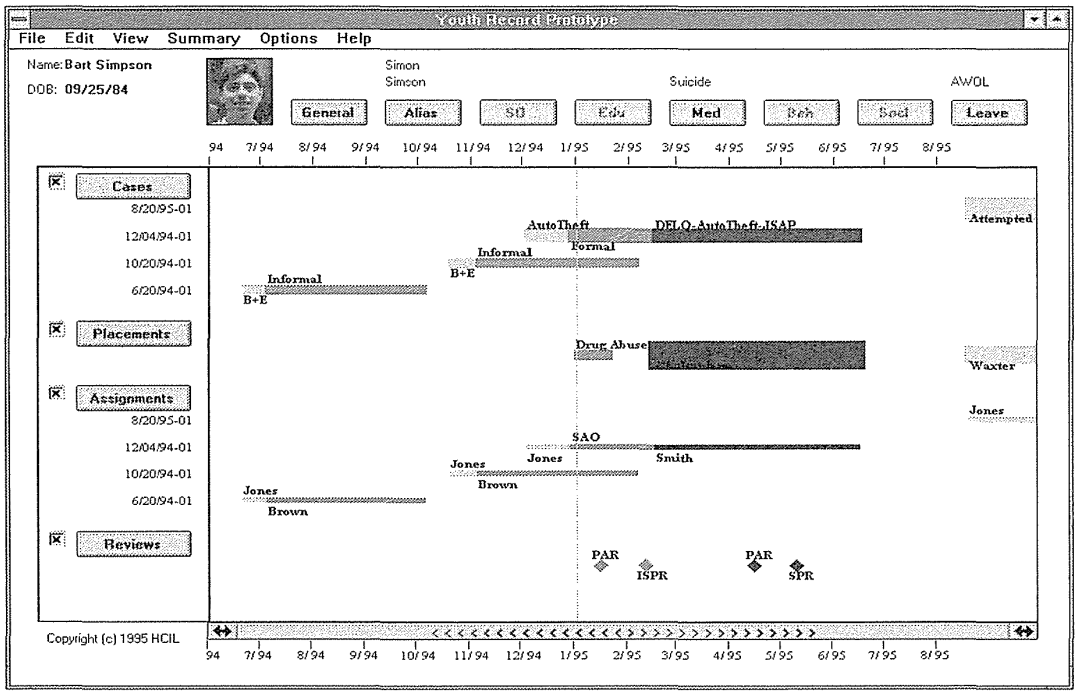


Figure 1.5

Youth Record prototype using the Lifelines display to show a case history for the Maryland Department of Juvenile Justice. (Used with permission of the University of Maryland Human–Computer Interaction Laboratory, College Park, MD.)

4). User-interface building tools (see Chapter 5) enable rapid implementation and easy revision. Evaluation of designs refines the understanding of appropriateness for each choice.

Successful designers go beyond the vague notion of "user friendliness," probing deeper than simply making a checklist of subjective guidelines. They have a thorough understanding of the diverse community of users and the tasks that must be accomplished. Moreover, they are deeply committed to serving the users, which strengthens their resolve when they face the pressures of short deadlines, tight budgets, and weak-willed compromisers.

Effective systems generate positive feelings of success, competence, mastery, and clarity in the user community. The users are not encumbered by the computer and can predict what will happen in response to each of their actions. When an interactive system is well designed, the interface almost disappears, enabling users to concentrate on their work, exploration, or pleasure. Creating an environment in which tasks are carried out almost effortlessly and users are "in the flow" requires a great deal of hard work from the designer.

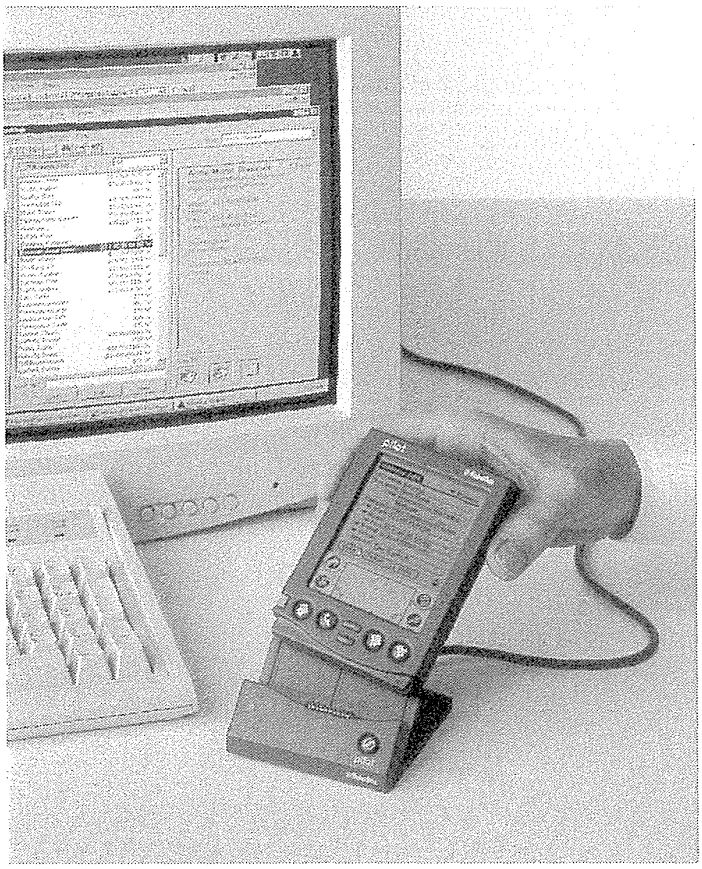


Figure 1.6

U.S. Robotics Pilot portable computer. The convenient docking station allows easy synchronization of files with a desktop computer. (Used with permission of U.S. Robotics.)

Setting explicit goals helps designers to achieve those goals. In getting beyond the vague quest for user-friendly systems, managers and designers can focus on specific goals that include well-defined system-engineering and measurable human-factors objectives. The U.S. Military Standard for Human Engineering Design Criteria (1989) states these purposes:

- Achieve required performance by operator, control, and maintenance personnel
- Minimize skill and personnel requirements and training time
- Achieve required reliability of personnel–equipment combinations
- Foster design standardization within and among systems

1.2.1 Proper functionality

The first step is to ascertain the necessary functionality—what tasks and subtasks must be carried out. The frequent tasks are easy to determine, but the occasional tasks, the exceptional tasks for emergency conditions, and the

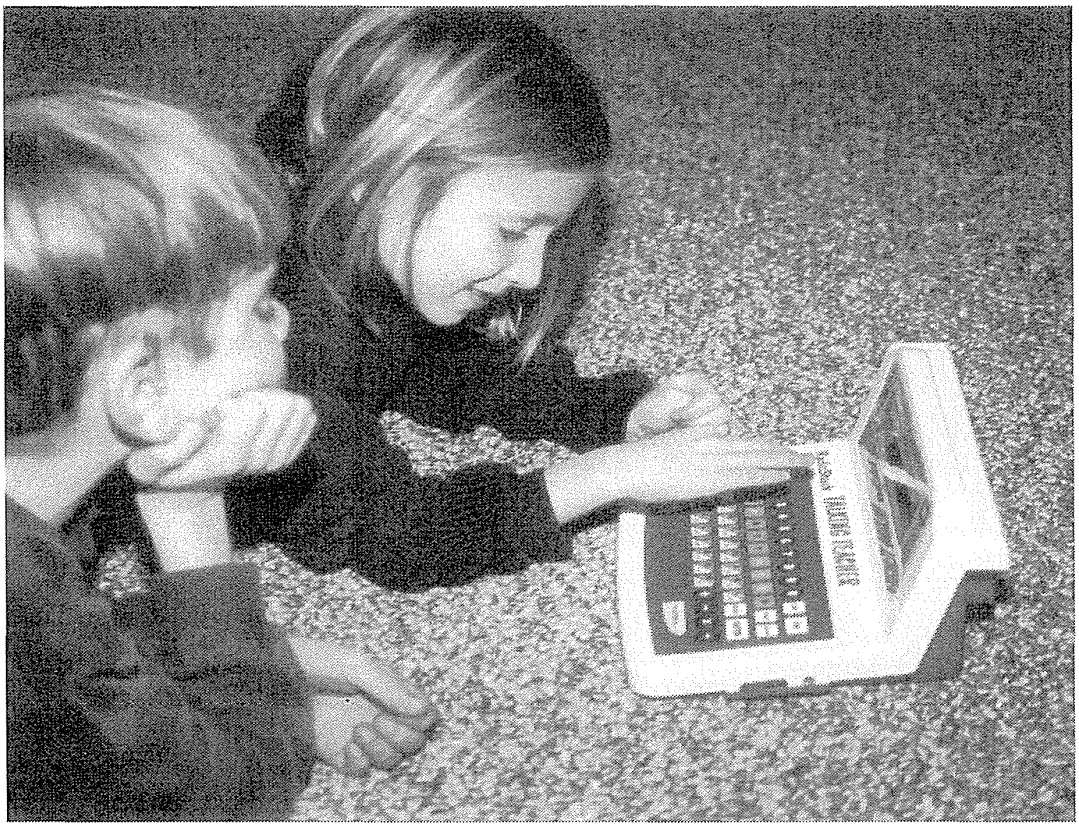


Figure 1.7

Children's educational game computer (Talking Teacher from Radio Shack, Tandy Corp., Ft. Worth, TX 76102), which has voice instructions and feedback with a one-line visual display. Games have three levels of difficulty and include word, number, and musical exercises.

repair tasks to cope with errors in use of the system are more difficult to discover. Task analysis is central, because systems with inadequate functionality frustrate the user and are often rejected or underutilized. If the functionality is inadequate, it does not matter how well the human interface is designed. Excessive functionality is also a danger, and providing it is probably the more common mistake of designers, because the clutter and complexity make implementation, maintenance, learning, and usage more difficult.

1.2.2 Reliability, availability, security, and data integrity

A vital second step is ensuring proper system reliability: commands must function as specified, displayed data must reflect the database contents, and updates must be applied correctly. Users' trust of systems is fragile; one

experience with misleading data or unexpected results will undermine for a long time a person's willingness to use a system. The software architecture, hardware components, and network support must ensure high availability. If the system is not available or introduces errors, then it does not matter how well the human interface is designed. Designers also must pay attention to ensuring privacy, security, and data integrity. Protection must be provided from unauthorized access, inadvertent destruction of data, or malicious tampering.

1.2.3 Standardization, integration, consistency, and portability

As the number of users and software packages increases, the pressures for and benefits of standardization grow. Slight differences among systems not only increase learning times, but also can lead to annoying and dangerous errors. Gross differences among systems require substantial retraining and burden users in many ways. Incompatible storage formats, hardware, and software versions cause frustration, inefficiency, and delay. Designers must decide whether the improvements they offer are useful enough to offset the disruption to the users.

Standardization refers to common user-interface features across multiple applications. Apple Computers (1987) successfully developed an early standard that was widely applied by thousands of developers, enabling users to learn multiple applications quickly. IBM's Common User Access (1989, 1991, 1993) specifications came later; and when the Microsoft Windows (1995) interface became standardized, it became a powerful force.

Integration across application packages and software tools was one of the key design principles in Unix. (Portability across hardware platforms was another.) The command language was standard from the beginning (with some divergences), but there are now competing graphical user interfaces (GUIs), many built around the X and Motif standards.

Consistency primarily refers to common action sequences, terms, units, layouts, color, typography, and so on within an application program. Consistency is a strong determinant of success of systems. It is naturally extended to include compatibility across application programs and compatibility with paper or non–computer-based systems. Compatibility across versions is a troubling demand, since the desire to accommodate novel functionality or improved designs competes with the benefits of consistency.

Portability refers to the potential to convert data and to share user interfaces across multiple software and hardware environments. Arranging for portability is a challenge for designers who must contend with different display sizes and resolutions, color capabilities, pointing devices, data formats, and so on. Some user-interface building tools help by generating code for Macintosh, Windows, OS/2, Unix, and other environments so that the interfaces are similar in each environment or resemble the style in those environments. Standard

text files (in ASCII) can be moved easily across environments, but graphic images, spreadsheets, video images, and so on are more difficult to convert.

1.2.4 Schedules and budgets

Careful planning and courageous management are needed if a project is to be completed on schedule and within budget. Delayed delivery or cost overruns can threaten a system because of the confrontational political atmosphere in a company, or because the competitive market environment contains potentially overwhelming forces. If an in-house system is delivered late, then other projects are affected, and the disruption may cause managers to choose to install an alternative system. If a commercial system is too costly, customer resistance may emerge to prevent widespread acceptance, allowing competitors to capture the market.

Proper attention to human-factors principles and rigorous testing often leads to reduced cost and rapid development. A carefully tested design generates fewer changes during implementation and avoids costly updates after release. The business case for human factors in computer and information systems is strong (Klemmer, 1989; Chapanis, 1991; Landauer, 1995), as demonstrated by many successful products whose advantage lay in their superior user interfaces.

1.3 Goals of User-Interface Design

If adequate functionality has been chosen, reliability is ensured, standardization addressed and schedule plus budgetary planning is complete, then developers can focus their attention on the design and testing process. The multiple design alternatives must be evaluated for specific user communities and for specific benchmark tasks. A clever design for one community of users may be inappropriate for another community. An efficient design for one class of tasks may be inefficient for another class.

The relativity of design played a central role in the evolution of information services at the Library of Congress (Marchionini et al., 1993). Two of the major uses of computer systems were cataloging new books and searching the online book catalog. Separate systems for these tasks were created that optimized the design for one task and made the complementary task difficult. It would be impossible to say which was better because both were fine systems, but they were serving different needs. Posing such a question would be like asking whether the New York Philharmonic Orchestra was better than the New York Yankees baseball team.

The situation became even more complex when Congressional staffers and then the public were invited to use the search systems. Three- to six-hour

training courses were appropriate for Congressional staffers, but the first-time public users were overwhelmed by the command language and complex cataloging rules. Eventually a touchscreen interface with reduced functionality and better information presentation was developed and became a big success in the public reading rooms. The next step in evolution was the development of a World Wide Web version of the catalog to allow users anywhere in the world to access the catalog and other databases. These changing user communities and requirements each led to interface changes, even though the database and services remained similar.

Careful determination of the user community and of the benchmark set of tasks is the basis for establishing human-factors goals. For each user and each task, precise measurable objectives guide the designer, evaluator, purchaser, or manager. These five measurable human factors are central to evaluation:

1. *Time to learn* How long does it take for typical members of the user community to learn how to use the commands relevant to a set of tasks?
2. *Speed of performance* How long does it take to carry out the benchmark tasks?
3. *Rate of errors by users* How many and what kinds of errors do people make in carrying out the benchmark tasks? Although time to make and correct errors might be incorporated into the speed of performance, error handling is such a critical component of system usage that it deserves extensive study.
4. *Retention over time* How well do users maintain their knowledge after an hour, a day, or a week? Retention may be linked closely to time to learn, and frequency of use plays an important role.
5. *Subjective satisfaction* How much did users like using various aspects of the system? The answer can be ascertained by interview or by written surveys that include satisfaction scales and space for free-form comments.

Every designer would like to succeed in every category, but there are often forced tradeoffs. If lengthy learning is permitted, then task-performance times may be reduced by use of complex abbreviations, macros, and shortcuts. If the rate of errors is to be kept extremely low, then speed of performance may have to be sacrificed. In some applications, subjective satisfaction may be the key determinant of success; in others, short learning times or rapid performance may be paramount. Project managers and designers must be aware of the tradeoffs and must make their choices explicit and public. Requirements documents and marketing brochures should make clear which goals are primary.

After multiple design alternatives are raised, the leading possibilities should be reviewed by designers and users. Low-fidelity paper mockups are

useful, but high-fidelity online prototypes create a more realistic environment for review. Design teams negotiate the guidelines document to make explicit the permissible formats, sequences, terminology, and so on. Then, the interface design is created with suitable prototyping tools, and testing can begin to ensure that the user-interface design goals are met. The user manual and the technical reference manual can be written before the implementation to provide another review and perspective on the design. Next, the implementation can be carried out with proper software tools; this task should be a modest one if the design is complete and precise. Finally, the acceptance test certifies that the delivered system meets the goals of the designers and customers. The development and evaluation process is described in greater detail in Chapters 3 and 4.

1.4 Motivations for Human Factors in Design

The enormous interest in human factors of interactive systems arises from the complementary recognition of how poorly designed many current systems are and of how genuinely developers desire to create elegant systems that serve the users effectively. This increased concern emanates from four primary sources: life-critical systems; industrial and commercial uses; office, home, and entertainment applications; and exploratory, creative, and collaborative systems.

1.4.1 Life-critical systems

Life-critical systems include those that control air traffic, nuclear reactors, power utilities, staffed spacecraft, police or fire dispatch, military operations, and medical instruments. In these applications high costs are expected, but they should yield high reliability and effectiveness. Lengthy training periods are acceptable to obtain rapid, error-free performance, even when the users are under stress. Subjective satisfaction is less of an issue because the users are well motivated. Retention is obtained by frequent use of common functions and practice sessions for emergency actions.

1.4.2 Industrial and commercial uses

Typical industrial and commercial uses include banking, insurance, order entry, inventory management, airline and hotel reservations, car rentals, utility billing, credit-card management, and point-of-sales terminals. In these cases, costs shape many judgments; lower cost may be preferred even if there is some sacrifice in reliability. Operator training time is expensive, so ease of

learning is important. The tradeoffs for speed of performance and error rates are governed by the total cost over the system's lifetime. Subjective satisfaction is of modest importance; retention is obtained by frequent use. Speed of performance becomes central for most of these applications because of the high volume of transactions, but operator fatigue or burnout is a legitimate concern. Trimming 10 percent off the mean transaction time means 10-percent fewer operators, 10-percent fewer terminal workstations, and possibly a 10-percent reduction in hardware costs. A study by developers of a system to manage telephone directory assistance indicated that a 0.8-second reduction in the 15-second mean time per call would save $40 million per year (Springer, 1987).

1.4.3 Office, home, and entertainment applications

The rapid expansion of office, home, and entertainment applications is the third source of interest in human factors. Personal-computing applications include word processing, automated transaction machines, video games, educational packages, information retrieval, electronic mail, computer conferencing, and small-business management. For these systems, ease of learning, low error rates, and subjective satisfaction are paramount because use is frequently discretionary and competition is fierce. If the users cannot succeed quickly, they will abandon the use of a computer or try a competing package. In cases where use is intermittent, retention is likely to be faulty, so online assistance becomes important.

Choosing the right functionality is difficult. Novices are best served by a constrained simple set of actions; but as users' experience increases, so does their desire for more extensive functionality and rapid performance. A layered or level-structured design is one approach to graceful evolution from novice to expert usage. Low cost is important because of lively competition, but extensive design and testing can be amortized over the large number of users.

1.4.4 Exploratory, creative, and cooperative systems

An increasing fraction of computer use is dedicated to supporting human intellectual and creative enterprises. Electronic encyclopedias, World Wide Web browsing, collaborative writing, statistical hypothesis formation, business decision making, and graphical presentation of scientific simulation results are examples of exploratory environments. Creative environments include writer's toolkits or workbenches, architecture or automobile design systems, artist or programmer workstations, and music-composition systems. Decision-support tools aid knowledgeable users in medical diagnosis, finance, industrial-process management, satellite-orbit determination, and military command and control. Cooperative systems enable two or more people to work together, even if the users are separated by time and space, through use

of electronic text, voice, and video mail; through electronic meeting systems that facilitate face-to-face meetings; or through groupware that enables remote collaborators to work concurrently on a document, spreadsheet, or image.

In these systems, the users may be knowledgeable in the task domain but novices in the underlying computer concepts. Their motivation is often high, but so are their expectations. Benchmark tasks are more difficult to describe because of the exploratory nature of these applications. Usage can range from occasional to frequent. In short, it is difficult to design and evaluate these systems. At best, designers can pursue the goal of having the computer vanish as users become completely absorbed in their task domain. This goal seems to be met most effectively when the computer provides a direct-manipulation representation of the world of action. Then, tasks are carried out by rapid familiar selections or gestures, with immediate feedback and a new set of choices.

1.5 Accommodation of Human Diversity

The remarkable diversity of human abilities, backgrounds, motivations, personalities, and workstyles challenges interactive-system designers. A right-handed female designer with computer training and a desire for rapid interaction using densely packed screens may have a hard time developing a successful workstation for left-handed male artists with a more leisurely and freeform workstyle. Understanding the physical, intellectual, and personality differences among users is vital.

1.5.1 Physical abilities and physical workplaces

Accommodating the diverse human perceptual, cognitive, and motor abilities is a challenge to every designer. Fortunately, there is much literature reporting research and experience from design projects with automobiles, aircraft, typewriters, home appliances, and so on that can be applied to the design of interactive computer systems. In a sense, the presence of a computer is only incidental to the design; human needs and abilities are the guiding forces.

Basic data about human dimensions comes from research in *anthropometry* (Dreyfus, 1967; Roebuck et al., 1975). Thousands of measures of hundreds of features of people—male and female, young and adult, European and Asian, underweight and overweight, and tall and short—provide data to construct means and 5- to 95-percentile groupings. Head, mouth, nose, neck, shoulder, chest, arm, hand, finger, leg, and foot sizes have been carefully cataloged for a variety of populations. The great diversity in these static measures reminds

us that there can be no image of an "average" user, and that compromises must be made or multiple versions of a system must be constructed.

The choice of keyboard design parameters (see Section 9.2) evolved to meet the physical abilities of users in terms of distance between keys, size of keys, and required pressure. People with especially large or small hands may have difficulty in using standard keyboards, but a substantial fraction of the population is well served by one design. On the other hand, since screen-brightness preferences vary substantially, designers must provide a knob to enable user control. Controls for chair seat and back heights, or for display-screen angles, also allow individual adjustment. When a single design cannot accommodate a large fraction of the population, then multiple versions or adjustment controls are helpful.

Physical measures of static human dimensions are not enough. Measures of dynamic actions—such as reach distance while seated, speed of finger presses, or strength of lifting—are also necessary (Bailey, 1996).

Since so much of work is related to perception, designers need to be aware of the ranges of human perceptual abilities (Schiff, 1980). Vision is especially important and has been thoroughly studied (Wickens, 1992). For example, researchers consider human response time to varying visual stimuli, or time to adapt to low or bright light. They examine human capacity to identify an object in context, or to determine the velocity or direction of a moving point. The visual system responds differently to various colors, and some people are color blind. People's spectral range and sensitivity vary. Peripheral vision is quite different from perception of images in the fovea. Flicker, contrast, and motion sensitivity must be considered, as must the impact of glare and of visual fatigue. Depth perception, which allows three-dimensional viewing, is based on several cues. Some viewing angles and distances make the screen easier to read. Finally, designers must consider the needs of people who have eye disorders, damage, or disease, or who wear corrective lenses.

Other senses are also important: touch for keyboard or touchscreen entry, and hearing for audible cues, tones, and speech input or output (see Chapter 9). Pain, temperature sensitivity, taste, and smell are rarely used for input or output in interactive systems, but there is room for imaginative applications.

These physical abilities influence elements of the interactive-system design. They also play a prominent role in the design of the workplace or workstation (or playstation). The American National Standard for Human Factors Engineering of Visual Display Terminal Workstations (1988) lists these concerns:

- Work-surface and display support height
- Clearance under work surface for legs
- Work-surface width and depth
- Adjustability of heights and angles for chairs and work surfaces
- Posture—seating depth and angle; back-rest height and lumbar support

- Availability of armrests, footrests, and palmrests
- Use of chair casters

Workplace design is important in ensuring high job satisfaction, high performance, and low error rates. Incorrect table heights, uncomfortable chairs, or inadequate space to place documents can substantially impede work. The Standard document also addresses such issues as illumination levels (200 to 500 lux); glare reduction (antiglare coatings, baffles, mesh, positioning); luminance balance and flicker; equipment reflectivity; acoustic noise and vibration; air temperature, movement, and humidity; and equipment temperature.

The most elegant screen design can be compromised by a noisy environment, poor lighting, or a stuffy room, and that compromise will eventually lower performance, raises error rates, and discourage even motivated users.

Another physical-environment consideration involves room layout and the sociology of human interaction. With multiple workstations for a classroom or office, alternate layouts can encourage or limit social interaction, cooperative work, and assistance with problems. Because users can often quickly help one another with minor problems, there may be an advantage to layouts that group several terminals close together or that enable supervisors or teachers to view all screens at once from behind. On the other hand, programmers, reservations clerks, or artists may appreciate the quiet and privacy of their own workspace.

The physical design of workplaces is often discussed under the term *ergonomics*. Anthropometry, sociology, industrial psychology, organizational behavior, and anthropology may offer useful insights in this area.

1.5.2 Cognitive and perceptual abilities

A vital foundation for interactive-systems designers is an understanding of the cognitive and perceptual abilities of the users (Kantowitz and Sorkin, 1983; Wickens, 1992). The human ability to interpret sensory input rapidly and to initiate complex actions makes modern computer systems possible. In milliseconds, users recognize slight changes on their displays and begin to issue a stream of commands. The journal *Ergonomics Abstracts* offers this classification of human *cognitive processes*:

- Short-term memory
- Long-term memory and learning
- Problem solving
- Decision making
- Attention and set (scope of concern)
- Search and scanning
- Time perception

They also suggest this set of *factors affecting perceptual and motor performance*:

- Arousal and vigilance
- Fatigue
- Perceptual (mental) load
- Knowledge of results
- Monotony and boredom
- Sensory deprivation
- Sleep deprivation
- Anxiety and fear
- Isolation
- Aging
- Drugs and alcohol
- Circadian rhythms

These vital issues are not discussed in depth in this book, but they have a profound influence on the quality of the design of most interactive systems. The term *intelligence* is not included in this list, because its nature is controversial and measuring pure intelligence is difficult.

In any application, background experience and knowledge in the task domain and the interface domain (see Section 2.2) play key roles in learning and performance. Task- or computer-skill inventories can be helpful in predicting performance.

1.5.3 Personality differences

Some people dislike computers or are made anxious by them; others are attracted to or are eager to use computers. Often, members of these divergent groups disapprove or are suspicious of members of the other community. Even people who enjoy using computers may have very different preferences for interaction styles, pace of interaction, graphics versus tabular presentations, dense versus sparse data presentation, step-by-step work versus all-at-once work, and so on. These differences are important. A clear understanding of personality and cognitive styles can be helpful in designing systems for a specific community of users.

A fundamental difference is one between men and women, but no clear pattern of preferences has been documented. It is often pointed out that the preponderance of video-arcade game players and designers are young males. There are women players of any game, but popular choices among women for early videogames were Pacman and its variants, plus a few other games such as Donkey Kong or Tetris. We have only speculations regarding why women prefer these games. One female commentator labeled Pacman "oral

aggressive" and could appreciate the female style of play. Other women have identified the compulsive cleaning up of every dot as an attraction. These games are distinguished by their less violent action and sound track. Also, the board is fully visible, characters have personality, softer color patterns are used, and there is a sense of closure and completeness. Can these informal conjectures be converted to measurable criteria and then validated? Can designers become aware of the needs and desires of women, and create video games that will be more attractive to women than to men?

Turning from games to office automation, the largely male designers may not realize the effect on women users when the command names require the users to KILL a file or ABORT a program. These and other potential unfortunate mismatches between the user interface and the user might be avoided by more thoughtful attention to individual differences among users. Huff (1987) found a bias when he asked teachers to design educational games for boys or girls. The designers created gamelike challenges when they expected boys as users and used more conversational dialogs when they expected girls as users. When told to design for students, the designers produced boy-style games.

Unfortunately, there is no simple taxonomy of user personality types. A popular technique is to use the Myers–Briggs Type Indicator (MBTI) (Shneiderman, 1980), which is based on Carl Jung's theories of personality types. Jung conjectured that there were four dichotomies:

- *Extroversion versus introversion* Extroverts focus on external stimuli and like variety and action, whereas introverts prefer familiar patterns, rely on their inner ideas, and work alone contentedly.
- *Sensing versus intuition* Sensing types are attracted to established routines, are good at precise work and enjoy applying known skills, whereas intuitive types like solving new problems and discovering new relations but dislike taking time for precision.
- *Perceptive versus judging* Perceptive types like to learn about new situations, but may have trouble making decisions, whereas judging types like to make a careful plan and will seek to carry through the plan even if new facts change the goal.
- *Feeling versus thinking* Feeling types are aware of other people's feelings, seek to please others and relate well to most people, whereas thinking types are unemotional, may treat people impersonally and like to put things in logical order.

The theory behind the MBTI provides portraits of the relationships between professions and personality types and between people of different personality types. It has been applied to testing of user communities and has provided guidance for designers.

Many hundreds of psychological scales have been developed, including risk taking versus risk avoidance; internal versus external locus of control; reflective versus impulsive behavior; convergent versus divergent thinking; high versus low anxiety; tolerance for stress; tolerance for ambiguity, motivation, or compulsiveness; field dependence versus independence; assertive versus passive personality; and left- versus right-brain orientation. As designers explore computer applications for home, education, art, music, and entertainment, they will benefit from paying greater attention to personality types.

1.5.4 Cultural and international diversity

Another perspective on individual differences has to do with cultural, ethnic, racial, or linguistic background (Fernandes, 1995). It seems obvious that users who were raised learning to read Japanese or Chinese will scan a screen differently from users who were raised learning to read English or French. Users from cultures that have a more reflective style or respect for ancestral traditions may prefer interfaces different from those chosen by users from cultures that are more action oriented or novelty based.

Little is known about computer users from different cultures, but designers are regularly called on to make designs for other languages and cultures. The growth of a worldwide computer market (many U.S. companies have more than one-half of their sales in overseas markets) means that designers must prepare for internationalization. Software architectures that facilitate customization of local versions of user interfaces should be emphasized. For example, all text (instructions, help, error messages, labels) might be stored in files, so that versions in other languages could be generated with no or little additional programming. Hardware concerns include character sets, keyboards, and special input devices. User-interface design concerns for internationalization include the following:

- Characters, numerals, special characters, and diacriticals
- Left-to-right versus right-to-left versus vertical input and reading
- Date and time formats
- Numeric and currency formats
- Weights and measures
- Telephone numbers and addresses
- Names and titles (Mr., Ms., Mme., M., Dr.)
- Social-security, national identification, and passport numbers
- Capitalization and punctuation

- Sorting sequences
- Icons, buttons, colors
- Pluralization, grammar, spelling
- Etiquette, policies, tone, formality, metaphors

The list is long and yet incomplete. Whereas early designers were often excused from cultural and linguistic slips, the current highly competitive atmosphere means that more effective localization will often produce a strong advantage. To promote effective designs, companies should run usability studies with users from each country, culture, and language community (Nielsen, 1990).

1.5.5 Users with disabilities

The flexibility of computer software makes it possible for designers to provide special services to users who have disabilities (Edwards, 1995; McWilliams, 1984; Glinert and York, 1992). The U.S. General Services Administration's (GSA) guide, *Managing End User Computing for Users with Disabilities* (1991), describes effective accommodations for users who have low vision or are blind, users who have hearing impairments, and users who have mobility impairments. Enlarging portions of a display (Kline and Glinert, 1995) or converting displays to braille or voice output (Durre and Glander, 1991) can be done with hardware and software supplied by many vendors. Text-to-speech conversion can help blind users to receive electronic mail or to read text files, and speech-recognition devices permit voice-controlled operation of some software. Graphical user interfaces were a setback for vision-impaired users, but technology innovations facilitate conversion of spatial information into nonvisual modes (Poll and Waterham, 1995; Thatcher, 1994; Mynatt and Weber, 1994).

Users with hearing impairments often can use computers with only simple change (conversion of tones to visual signals is often easy to accomplish), and can benefit from office environments that make heavy use of electronic mail and facsimile transmission (FAX). Telecommunications devices for the deaf (TDD) enable telephone access to information (such as train or airplane schedules) and services (federal agencies and many companies offer TDD access). Special input devices for users with physical disabilities will depend on the user's specific impairment; numerous assisting devices are available. Speech recognition, eye-gaze control, head-mounted optical mouse, and many other innovative devices (even the telephone) were pioneered for the needs of disabled users (see Chapter 9).

Designers can benefit by planning early to accommodate users who have disabilities, since substantial improvements can be made at low or no cost. The term *computer curbcuts* brings up the image of sidewalk cutouts to permit wheelchair access that are cheaper to build than standard curbs if they are planned rather than added later. Similarly, moving the on–off switch to the front of a computer adds a minimal change to the cost of manufacturing and helps mobility-impaired users, as well as other users. The motivation to accommodate users who have disabilities has increased since the enactment of U.S. Public Laws 99–506 and 100–542, which require U.S. government agencies to establish accessible information environments that accommodate employees and citizens who have disabilities. Any company wishing to sell products to the U.S. government should adhere to the GSA recommendations (1991). Further information about accommodation in workplaces, schools, and the home is available from many sources:

- Private foundations (e.g., the American Foundation for the Blind)
- Associations (e.g., the Alexander Graham Bell Association for the Deaf, the National Association for the Deaf, and the Blinded Veterans Association)
- Government agencies (e.g., the National Library Service for the Blind and Physically Handicapped of the Library of Congress and the Center for Technology in Human Disabilities at the Maryland Rehabilitation Center)
- University groups (e.g., the Trace Research and Development Center on Communications and the Control and Computer Access for Handicapped Individuals at the University of Wisconsin)
- Manufacturers (e.g., Apple, AT&T, DEC, and IBM)

Learning-disabled children account for two percent of the school-age population in the United States. Their education can be positively influenced by design of special courseware with limits on lengthy textual instructions, confusing graphics, extensive typing, and difficult presentation formats (Neuman, 1991). Based on observations of 62 students using 26 packages over 5.5 months, Neuman's advice to designers of courseware for learning-disabled students is applicable to all users:

- Present procedures, directions, and verbal content at levels and in formats that make them accessible even to poor readers.
- Ensure that response requirements do not allow students to complete programs without engaging with target concepts.
- Design feedback sequences that explain the reasons for students' errors and that lead students through the processes necessary for responding correctly.

- Incorporate reinforcement techniques that capitalize on students' sophistication with out-of-school electronic materials.

Our studies with minimally learning-disabled fourth, fifth, and sixth graders learning to use word processors reinforce the need for direct manipulation (see Chapter 6) of visible objects of interest (MacArthur and Shneiderman, 1986). The potential for great benefit to people with disabilities is one of the unfolding gifts of computing. The Association for Computing Machinery (ACM) Special Interest Group on Computers and the Physically Handicapped (SIGCAPH) publishes a quarterly newsletter of interest to workers in this area and runs the annual conference on Assistive Technology (ASSETS).

1.5.6 Elderly users

Most people grow old. There can be many pleasures and satisfactions to seniority, but there are also negative physical, cognitive, and social consequences of aging. Understanding the human factors of aging can lead us to computer designs that will facilitate access by the elderly. The benefits to the elderly include meeting practical needs for writing, accounting, and the full range of computer tools, plus the satisfactions of education, entertainment, social interaction, communication, and challenge (Furlong and Kearsley, 1990). Other benefits include increased access of the society to the elderly for their experience, increased participation of the elderly in society through communication networks, and improved chances for productive employment of the elderly.

The National Research Council's report on Human Factors Research Needs for an Aging Population describes aging as

> A nonuniform set of progressive changes in physiological and psychological functioning.... Average visual and auditory acuity decline considerably with age, as do average strength and speed of response.... [People experience] loss of at least some kinds of memory function, declines in perceptual flexibility, slowing of "stimulus encoding," and increased difficulty in the acquisition of complex mental skills,... visual functions such as static visual acuity, dark adaptation, accommodation, contrast sensitivity, and peripheral vision decline, on average, with age. (Czaja, 1987)

This list has its discouraging side, but many people experience only modest effects and continue participating in many activities, even through their nineties.

The further good news is that computer-systems designers can do much to accommodate elderly users, and thus to give the elderly access to the ben-

eficial aspects of computing and network communication. How many young people's lives might be enriched by electronic-mail access to grandparents or great-grandparents? How many businesses might benefit from electronic consultations with experienced senior citizens? How many government agencies, universities, medical centers, or law firms could advance their goals by easily available contact with knowledgeable elderly citizens? As a society, how might we all benefit from the continued creative work of senior citizens in literature, art, music, science, or philosophy?

As the U.S. population grows older, designers in many fields are adapting their work to serve the elderly. Larger street signs, brighter traffic lights, and better nighttime lighting can make driving safer for drivers and pedestrians. Similarly, larger fonts, higher display contrast, easier-to-use pointing devices, louder audio tones, and simpler command languages are just a few of the steps that user-interface designers can take to improve access for the elderly (Tobias, 1987; Christiansen et al., 1989). Many of these adjustments can be made through software-based control panels that enable users to tailor the system to their changing personal needs. System developers have yet to venture actively into the potentially profitable world of golden-age software, in parallel to the growing market in kidware. Let's do it *before* Bill Gates turns 65!

Electronic-networking projects, such as the San Francisco–based SeniorNet, are exploring the needs of elderly users (anyone over 55 years of age may join) for computing services, networking, and training. Computer games are also attractive for the elderly because they stimulate social interaction, provide practice in sensorimotor skills such as eye–hand coordination, enhance dexterity, and improve reaction time. In addition, meeting a challenge and gaining a sense of accomplishment and mastery are helpful in improving self-image for anyone (Whitcomb, 1990).

In our research group's brief experiences in bringing computing to two residences for elderly people, we also found that the users' widespread fear of computers and belief that they were incapable of using computers gave way quickly with a few positive experiences. These elderly users, who explored video games, word processors, and educational games, felt quite satisfied with themselves, were eager to learn more, and transferred their new-found enthusiasm to trying automated bank machines or supermarket touchscreen computers. Suggestions for redesign to meet the needs of elderly users (and possibly other users) emerged, such as the appeal of high-precision touchscreens compared with the mouse (see Chapter 9).

In summary, computing for elderly users provides an opportunity for the elderly, for system developers, and for all society. The Human Factors & Ergonomics Society has a Technical Group on Aging that publishes a newsletter at least twice a year and organizes sessions at conferences.

1.6 Goals for Our Profession

Clear goals are useful not only for system development but also for educational and professional enterprises. Three broad goals seem attainable: (1) influencing academic and industrial researchers; (2) providing tools, techniques, and knowledge for commercial systems implementors; and (3) raising the computer consciousness of the general public.

1.6.1 Influencing academic and industrial researchers

Early research in human–computer interaction was done largely by introspection and intuition, but this approach suffered from lack of validity, generality, and precision. The techniques of controlled psychologically-oriented experimentation can lead to a deeper understanding of the fundamental principles of human interaction with computers.

The reductionist scientific method has this basic outline:

- Understanding of a practical problem and related theory
- Lucid statement of a testable hypothesis
- Manipulation of a small number of independent variables
- Measurement of specific dependent variables
- Careful selection and assignment of subjects
- Control for bias in subjects, procedures, and materials
- Application of statistical tests
- Interpretation of results, refinement of theory, and guidance for experimenters

Materials and methods must be tested by pilot experiments, and results must be validated by replication in variant situations.

Of course, the highly developed and structured method of controlled experimentation has its weaknesses. It may be difficult or expensive to find adequate subjects, and laboratory conditions may distort the situation so much that the conclusions have no application. When we arrive at results for large groups of subjects by statistical aggregation, extremely good or poor performance by individuals may be overlooked. Furthermore, anecdotal evidence or individual insights may be given too little emphasis because of the authoritative influence of statistics.

In spite of these concerns, controlled experimentation provides a productive basis that can be modified to suit the situation. Anecdotal experiences and subjective reactions should be recorded, thinking aloud or protocol

approaches should be employed, field or case studies with extensive performance data collection should be carried out, and the individual insights of researchers, designers, and experimental participants should be captured.

Within computer science, there is a growing awareness of the need for greater attention to human-factors issues. Researchers who propose new programming languages or data-structure constructs are more aware of the need to match human cognitive skills. Developers of advanced graphics systems, agile manufacturing equipment, or computer-assisted design systems increasingly recognize that the success of their proposals depends on the construction of a suitable human interface. Researchers in these and other areas are making efforts to understand and measure human performance.

There is a grand opportunity to apply the knowledge and techniques of traditional psychology (and of recent subfields such as cognitive psychology) to the study of human–computer interaction. Psychologists are investigating human problem solving with computers to gain an understanding of cognitive processes and memory structures. The benefit to psychology is great, but psychologists also have the golden opportunity to influence dramatically an important and widely used technology.

Researchers in information science, business and management, education, sociology, anthropology, and other disciplines are benefitting and contributing by their study of human–computer interaction (National Research Council, 1983; Marchionini and Sibert, 1991). There are so many fruitful directions for research that any list can be only a provocative starting point. Here are a few.

- *Reduced anxiety and fear of computer usage* Although computers are widely used, they still serve only a fraction of the population. Many otherwise competent people resist use of computers. Some elderly users avoid helpful computer-based devices, such as bank terminals or word processors, because they are anxious about—or even fearful of—breaking the computer or making an embarrassing mistake. Interviews with nonusers of computers would help us to determine the sources of this anxiety and to formulate design guidelines for alleviating the fear. Tests could be run to determine the effectiveness of the redesigned systems and of improved training procedures.
- *Graceful evolution* Although novices may begin their interactions with a computer by using menu selection, they may wish to evolve to faster or more powerful facilities. Methods are needed to smooth the transition from novice to knowledgeable user to expert. The differing requirements of novice and experts in prompting, error messages, online assistance, display complexity, locus of control, pacing, and informative feedback all need investigation. The design of control panels to support adaptation and evolution is also an open topic.

- *Specification and implementation of interaction* User interface building tools (Chapter 5) reduce implementation times by an order of magnitude when they match the task. There are still many situations in which extensive coding in procedural languages must be added. Specification languages have been proposed, but these are still a long way from being complete and useful. Advanced research on tools to aid interactive-systems designers and implementers might have substantial payoff in reducing costs and improving quality. Tools for World Wide Web designers to enable automatic conversion for different computers, screen sizes, or modem speeds could be substantially improved, thereby facilitating universal access.
- *Direct manipulation* Visual interfaces in which users operate on a representation of the objects of interest are extremely attractive (Chapter 6). Empirical studies would refine our understanding of what is an appropriate analogical or metaphorical representation, and of what is the role of rapid, incremental, reversible operations. Newer forms of direct manipulation—such as visual languages, spatial visualization, remote control, telepresence, and virtual reality—are further topics for research.
- *Input devices* The plethora of input devices presents opportunities and challenges to system designers (Chapter 6). There are heated discussions about the relative merits of the high-precision touchscreen; stylus, pen, voice, eye-gaze, and gestural input; the mouse; the dataglove; and the force-feedback joystick. Such conflicts could be resolved through extensive experimentation with multiple tasks and user communities. Underlying issues include speed, accuracy, fatigue, error correction, and subjective satisfaction.
- *Online assistance* Although many systems offer some help or tutorial information online, we have only limited understanding of what constitutes effective design for novices, knowledgeable users, and experts (Chapter 12). The role of these aids and of online user consultants could be studied to assess effects on user success and satisfaction. The goal of *just-in-time* (JIT) training is elusive, but appealing.
- *Information exploration* As navigation, browsing, and searching of multimedia digital libraries and the World Wide Web become more common, the pressure for more effective strategies and tools will increase (Chapter 15). Users will want to filter, select, and restructure their information rapidly and with minimum effort, without fear of disorientation or of getting lost. Large databases of text, images, graphics, sound, and scientific data will become easier to explore with emerging information-visualization tools.

1.6.2 Providing tools, techniques, and knowledge for systems implementers

User-interface design and development are current hot topics, and international competition is lively. There is a great thirst for knowledge, software tools, design guidelines, and testing techniques. New user interface building tools (see Chapter 5) provide support for rapid prototyping and system development while aiding design consistency and simplifying evolutionary refinement.

Guidelines documents are being written for general audiences and for specific applications. Many projects are taking the productive route of writing their own guidelines, which are specifically tied to the problems of their application environment. These guidelines are constructed from experimental results, experience with existing non–computer-based systems, review of related computer-based systems, and knowledgeable guesswork.

Iterative usability studies and acceptance testing are appropriate during system development. Once the initial system is available, refinements can be made on the basis of online or printed surveys, individual or group interviews, or more controlled empirical tests of novel strategies (see Chapter 4).

Feedback from users during the development process and for evolutionary refinement can provide useful insights and guidance. Online electronic-mail facilities may allow users to send comments directly to the designers. Online user consultants and telephone hot-line workers can provide not only prompt assistance, but also much information about the activities and problems of the user community.

1.6.3 Raising the computer consciousness of the general public

The media are so filled with stories about computers that raising public consciousness of these tools may seem unnecessary. In fact, however, many people are still uncomfortable with computers. When they do finally use a bank machine or word processor, they may be fearful of making mistakes, anxious about damaging the equipment, worried about feeling incompetent, or threatened by the computer "being smarter than I am." These fears are generated, in part, by poor designs that have complex commands, hostile and vague error messages, tortuous and unfamiliar sequences of actions, or a deceptive anthropomorphic style.

One of my goals is to encourage users to translate their internal fears into outraged action. Instead of feeling guilty when they get a message such as `SYNTAX ERROR`, they should express their anger at the system designer who was so inconsiderate and thoughtless. Instead of feeling inadequate or

foolish because they cannot remember a complex sequence of commands, they should complain to the designer who did not provide a more convenient mechanism or should seek another product that does.

As examples of successful and satisfying systems become more visible, the crude designs will appear increasingly archaic and will become commercial failures. As designers improve interactive systems, some of these fears will recede and the positive experiences of competence, mastery, and satisfaction will flow in. Then, the images of computer scientists and of data-processing professionals will change in the public's view. The machine-oriented and technical image will give way to one of personal warmth, sensitivity, and concern for the user.

1.7 Practitioner's Summary

If you are designing an interactive system, a thorough task analysis can provide the information for a proper functional design. You should pay attention to reliability, availability, security, integrity, standardization, portability, integration, and the administrative issues of schedules and budgets. As design alternatives are proposed, they can be evaluated for their role in providing short learning times, rapid task performance, low error rates, ease of retention, and high user satisfaction. As the design is refined and implemented, you can test for accomplishment of these goals with pilot studies, expert reviews, usability tests, and acceptance tests. The rapidly growing literature and sets of design guidelines may be of assistance in developing your project standards and practices, and in accommodating the increasingly diverse and growing community of users.

1.8 Researcher's Agenda

The opportunities for researchers are unlimited. There are so many interesting, important, and doable projects that it may be hard to choose a direction. Each experiment has two parents: (1) the practical problems facing designers, and (2) the fundamental theories based on psychological principles of human behavior. Begin by proposing a lucid, testable hypothesis. Then consider the appropriate research methodology, conduct the experiment, collect the data, and analyze the results. Each experiment also has three children: (1) specific recommendations for the practical problem, (2) refinements of your theory of human performance, and (3) guidance to future experimenters. Each chapter of this book ends with specific research proposals.

World Wide Web Resources

WWW

This book is accompanied by an extensive website, prepared by Blaise Liffick (http://www.aw.com/DTUI), that includes pointers to additional resources tied to the contents of each chapter. In addition, this website contains information for instructors, students, practitioners, and researchers. The links for Chapter 1 include general resources on human-computer interaction, such as professional societies, government agencies, companies, bibliographies, and guidelines documents.

People seeking references to scientific journals and conferences now have an online bibliography for human–computer interaction. Built under the heroic leadership of Gary Perlman at Ohio State (perlman@turing.acm.org), it makes available almost 8000 journal, conference, and book abstracts. Some parts are searchable online, but most users FTP the files for personal use.

Three wonderful sets of pointers to World Wide Web resources are maintained by

1. Keith Instone (http://usableweb.com/hciel)
2. Hans de Graaf (http://is.twi.tudelft.nl/hci/)
3. Mikael Ericsson (http://www.ida.liu.se/labs/aslab/groups/um/hci/)

An excellent electronic mailing list (chi-announcements@acm.org) is maintained by SIGCHI. To subscribe, send electronic mail to `listserv@acm.org` with this line:

```
subscribe chi-announcements <your full name>.
```

Andrew Cohill (cohill@bev.net) maintains several listservs for the Human Factors & Ergonomics Society, including the lively CSTG-L. To subscribe, send electronic mail to listserv@listserv.vt.edu with this line:

```
subscribe cstg-L <your full name>.
```

http://www.aw.com/DTUI

References

Specialized references for this chapter appear here; general information resources are given in the section that follows immediately.

Chapanis, Alphonse, The business case for human factors in informatics. In Shackel, Brian and Richardson, Simon (Editors), *Human Factors for Informatics Usability*, Cambridge University Press, Cambridge, U.K. (1991), 39–71.

Christiansen, M., Chaudhary, S., Gottshall, R., Hartman, J., and Yatcilla, D., EASE: A user interface for the elderly. In Salvendy, G. and Smith, M. J. (Editors), *Designing and Using Human–Computer Interfaces and Knowledge Based Systems*, Elsevier Science Publishers B.V., Amsterdam, The Netherlands (1989), 428–435.

Czaja, Sara J. (Editor), *Human Factors Research Needs for an Aging Population*, National Academy Press, Washington, D.C. (1990).

Durre, Karl P. and Glander, Karl W., Design considerations for microcomputer based applications for the blind. In Nurminen, M. I., Jarvinen, P., and Weir, G. (Editors), *Human Jobs and Computer Interfaces*, North-Holland, Amsterdam, The Netherlands (1991).

Edwards, Alistair D.N., *Extra-Ordinary Human–Computer Interaction: Interfaces for Users with Disabilities*, Cambridge University Press, Cambridge, U.K. (1995).

Furlong, Mary and Kearsley, Greg, *Computers for Kids Over 60*, SeniorNet, San Francisco, CA (1990).

General Services Administration, Information Resources Management Services (GSI, IRMS), *Managing End User Computing for Users with Disabilities*, GSI, IRMS, Washington, D.C. (1991).

Glinert, Ephraim, P. and York, Bryant W., Computers and people with disabilities, *Communications of the ACM*, 35, 5 (May 1992), 32–35.

Huff, C. W. and Cooper, J., Sex bias in educational software: The effect of designers' stereotypes on the software they design, *Journal of Applied Social Psychology*, 17, 6 (June 1987), 519–532.

Kline, Richard L. and Glinert, Ephraim P., Improving GUI accessibility for people with low vision, *Proc. CHI' 95 Human Factors in Computer Systems*, ACM, New York (1995), 114–121.

MacArthur, Charles and Shneiderman, Ben, Learning disabled students' difficulties in learning to use a word processor: Implications for instruction and software evaluation, *Journal of Learning Disabilities*, 19, 4 (April 1986), 248–253.

Marchionini, Gary, Ashley, Maryle, and Korzendorfer, Lois, ACCESS at the Library of Congress, In Shneiderman, Ben (Editor), *Sparks of Innovation in Human–Computer Interaction*, Ablex Publishers, Norwood, NJ (1993), 251–258.

Marchionini, Gary and Sibert, John (Editors), An agenda for human–computer interaction: Science and engineering serving human needs, *ACM SIGCHI Bulletin* (October 1991), 17–32.

Mynatt, Elizabeth D. and Weber, Gerhard, Nonvisual presentation of graphical user interfaces: Contrasting two approaches, *CHI' 94 Human Factors in Computer Systems*, ACM, New York (1994), 166–172

National Research Council Committee on Human Factors, *Research Needs in Human Factors*, National Academy Press, Washington, D.C. (1983).

Neuman, Delia, Learning disabled students' interactions with commercial courseware: A naturalistic study, *Educational Technology Research and Development*, 39, 1 (1991), 31–49.

Poll, Leonard H. D. and Waterham, Ronald P., Graphical user interfaces and visually disabled users, *IEEE Transactions on Rehabilitation Engineering*, 3, 1 (March 1995), 65–69.

Springer, Carla J., Retrieval of information from complex alphanumeric displays: Screen formatting variables' effect on target identification time. In Salvendy, Gavriel (Editor), *Cognitive Engineering in the Design of Human–Computer Interaction and Expert Systems*, Elsevier, Amsterdam, The Netherlands (1987), 375–382.

Thatcher, James W., Screen Reader/2: Access to OS/2 and the graphical user interface, *Proc. ACM SIGCAPH—Computers and the Physically Handicapped, ASSETS '94* (1994), 39–47.

Tobias, Cynthia L., Computers and the elderly: A review of the literature and directions for future research, *Proc. Human Factors Society Thirty-First Annual Meeting*, Santa Monica, CA (1987), 866–870.

Whitcomb, G. Robert, Computer games for the elderly, *Proc. Conference on Computers and the Quality of Life '90*, ACM SIGCAS, New York (1990), 112–115.

General information resources

Primary journals include the following:

ACM Transactions on Computer–Human Interaction. Quarterly, ACM, 1515 Broadway, New York, NY 10036.

ACM Interactions: A Magazine for User Interface Designers. Quarterly, ACM, 1515 Broadway, New York, NY 10036.

Behaviour & Information Technology (BIT). Six times per year, Taylor & Francis Ltd, 4 John Street, London WCIN 2ET, U.K.

Human–Computer Interaction. Quarterly, Lawrence Erlbaum Associates, Inc., 365 Broadway, Hillsdale, NJ 07642.

Interacting with Computers. Quarterly, Butterworth Heinemann Ltd, Linacre House, Jordan Hill, Oxford OX2 8DP U.K.

International Journal of Human–Computer Studies, formerly *International Journal of Man–Machine Studies (IJMMS)*. Monthly, Academic Press, 24–28 Oval Road, London NW1 7DX, U.K.

International Journal of Human–Computer Interaction. Quarterly, Ablex Publishing Corporation, 355 Chestnut Street, Norwood, NJ 07648.

Other journals that regularly carry articles of interest are these:

ACM Computing Surveys

Communications of the ACM (CACM)

ACM Transactions on Graphics

ACM Transactions on Information Systems

Cognitive Science

Computer Supported Cooperative Work

Computers and Human Behavior

Ergonomics

Human Factors (HF)

Hypermedia

IEEE Computer
IEEE Computer Graphics and Applications
IEEE Software
IEEE Transactions on Systems, Man, and Cybernetics (IEEE SMC)
Journal of Visual Languages and Computing

The Association for Computing Machinery (ACM) has a Special Interest Group on Computer & Human Interaction (SIGCHI) that publishes a quarterly newsletter and holds regularly scheduled conferences. Other ACM Special Interest Groups such as Graphics (SIGGRAPH), Computers and the Physically Handicapped (SIGCAPH), and hypertext plus multimedia (SIGLINK) also cover this topic in their conferences and newsletters. The American Society for Information Science (ASIS) has a Special Interest Group on Human–Computer Interaction (SIGHCI) that publishes a quarterly newsletter and participates by organizing sessions at the annual ASIS convention. The International Federation for Information Processing has Technical Committee and Working Groups on human–computer interaction. The Human Factors & Ergonomics Society also has a Computer Systems Technical Group with a quarterly newsletter.

Conferences—such as the ones held by the ACM (the SIGCHI and SIGGRAPH especially), IEEE (the Visual Languages Symposium especially), ASIS, Human Factors & Ergonomics Society, and IFIP—often have relevant papers presented and published in the proceedings. The INTERACT, the Human–Computer Interaction International, and the Work with Display Units series of conferences (held approximately every other year) are also important resources with broad coverage of user-interface issues. Several more specialized ACM conferences may be of interest: User Interfaces Software and Technology, Hypertext, and Computer-Supported Cooperative Work.

The list of guidelines documents and books is a starting point to the large and growing literature in this area. Gerald Weinberg's 1971 book, *The Psychology of Computer Programming,* is a continuing inspiration to thinking about how people interact with computers. James Martin provided a thoughtful and useful survey of interactive systems in his 1973 book, *Design of Man–Computer Dialogues.* My 1980 book, *Software Psychology: Human Factors in Computer and Information Systems,* promoted the use of controlled experimental techniques and the reductionist scientific method. Rubinstein and Hersh, *The Human Factor: Designing Computer Systems for People* (1984), offered an appealing introduction and many useful guidelines. The first edition of this book, published in 1987, reviewed critical issues, offered guidelines for designers, and suggested research directions.

Don Norman's 1988 book, *The Psychology of Everyday Things,* is a refreshing look at the psychological issues in the design of the everyday technology that surrounds us. As a reader I was provoked equally by the sections dealing with doors or showers and computers or calculators. This book has a wonderful blend of levity and great depth of thinking, practical wisdom, and

thoughtful theory. A lively collection of essays was assembled in 1990 by Brenda Laurel in close collaboration with Apple, under the title *The Art of Human–Computer Interface Design*.

Recent recommended books are Hix and Hartson's 1993 *Developing User Interfaces*, Jakob Nielsen's 1993 *Usability Engineering*, Preece et al.'s 1994 *Human–Computer Interaction*, and Landauer's 1995 *The Trouble with Computers*. Two ambitious collections of papers appeared in 1995: Baecker et al.'s thoughtful and thorough commentaries enrich their 950 pages of reprints, and Perlman et al.'s careful selection of 79 papers on human–computer interaction from the Human Factors & Ergonomic Society conferences covers most topics.

An important development for the field was the creation (in late 1991) of a professional group, Usability Professionals Association (UPAdallas@aol.com), for usability testers, and a newsletter called *Common Ground*. The beginning of 1994 marked the appearance of ACM's professional magazine entitled *interactions*, and ACM's academic journal *Transactions on Computer–Human Interaction*.

Guidelines documents

General guidelines

American National Standard for Human Factors Engineering of Visual Display Terminal Workstations, ANSI/HFS Standard No. 100–1988, Human Factors Society, Santa Monica, CA (February 1988).

—Carefully considered standards for the design, installation, and use of visual display terminals. Emphasizes ergonomics and anthropometrics.

Engel, Stephen E. and Granda, Richard E., *Guidelines for Man/Display Interfaces*, Technical Report TR 00.2720, IBM, Poughkeepsie, NY (December 1975).

—An early and influential document that is the basis for several of the other guidelines documents.

Human Engineering Design Criteria for Military Systems, Equipment and Facilities, Military Standard MIL-STD–1472D, U.S. Government Printing Office, Washington, D.C. (March 14, 1989, and later changes).
ftp://archive.cis.ohiostate.edu/pub/hci/1472/

—Almost 300 pages (plus a 100-page index) covering traditional ergonometric or anthropometric issues. Later editions pay increasing attention to user–computer interfaces. Interesting and thought provoking, but sometimes outdated and difficult to read due to a six-level organization.

International Standards Organization, ISO 9241. Ergonomic Requirements for Office Work with Visual Display Terminals (VDT)s, Available from American National Standards Institute, 11 West 42nd Street, New York, NY.

—General introduction, dialogue principles, guidance on usability, presentation of information, user guidance, menu dialogues, command dialogues, direct manipulation dialogues, form filling dialogues.

NASA User-Interface Guidelines, Goddard Space Flight Center-Code 520, Greenbelt, MD (January 1996). http://groucho.gsfc.nasa.gov/Code_520/Code_522/Documents/HCI_Guidelines/

—The purpose of this document is to present user-interface guidelines that specifically address graphic and object-oriented interfaces operating in either distributed or independent systems environments. Principles and general guidelines are given, with many graphic-interface examples for a variety of platforms.

National Institute of Standards and Technology (NIST), *The User Interface Component of the Applications Portability Profile (FIPS PUB 158–1)*. Available from National Technical Information Service, U.S. Department of Commerce, Springfield, VA 22161.

—This standard is intended for use by computing professionals involved in system and application software development and implementation for network-based bitmapped graphic systems. This standard is part of a series of specifications needed for application portability. It covers the Data Stream Encoding, Data Stream Interface, and Subroutine Foundation layers of the reference model.

Smith, Sid L. and Mosier, Jane N., *Guidelines for Designing User Interface Software*, Report ESD-TR–86–278, Electronic Systems Division, MITRE Corporation, Bedford, MA (August 1986). Available from National Technical Information Service, Springfield, VA.

—This thorough document, which has undergone several revisions, begins with a good discussion of human-factors issues in design. It then covers data entry, data display, and sequence control. Guidelines are offered with comments, examples, exceptions, and references. This report is *the* place to start if you are creating your own guidelines.

Specific guidelines

Apple Human Interface Guidelines: The Apple Desktop Interface, Addison-Wesley, Reading, MA (1987), 144 pages.

—The Human Interface Group and the Technical Publications Group teamed up to produce this readable, example-filled book that starts with a thoughtful philosophy and then delves into precise details. It is required reading for anyone developing Macintosh software, and is an inspiration to people who are designing their own guidelines document; it also stimulates interesting reflections for researchers.

Apple Computer, Inc., *Macintosh Human Interface Guidelines*, Addison-Wesley, Reading, MA (1992), 384 pages.

—A major expansion of the previous citation, and a beautifully produced color book. A well-designed CD-ROM, *Making it Macintosh*, exemplifies these Mac guidelines, Addison-Wesley, Reading, MA (1993).

Bellcore, *Design Guide for Multiplatform Graphical User Interfaces LP-R13*, Bellare, Piscataway, NJ (December 1995).

—This document makes a diligent effort to provide guidance for designers of interfaces for implementation on several platforms, including Windows and Motif.

IBM, *Object-Oriented Interface Design: IBM Common User Access Guidelines*, Que Corp., Carmel, IN (December 1992), 708 pages.

—This book is the commercially published version of IBM's *CUA Guidelines.*

IBM Systems Application Architecture: Common User Access Guide to User Interface Design, IBM Document SC34–4289–00, (October 1991), 163 pages.

—This readable introduction to user-interface design is a textbook for software and user-interface designers that covers principles, components, and techniques.

IBM Systems Application Architecture: Common User Access Advanced Interface Design Reference, IBM Document SC34–4290–00, (October 1991), 401 pages.

—This volume is the latest version of IBM's *Guide* for application programmers who wish to adhere to the CUA design. It identifies what the interface components are and when to use them.

IBM System Application Architecture: Common User Access, Advanced Interface Design Guide, IBM Document SC26–4582–0, Boca Raton, FL (June 1989), 195 pages.

—This now-outdated version of the IBM standards shows progress over the 1987 document. It places heavy emphasis on graphic interaction, use of pointing devices, and windows. International standards for multiple languages are also given attention.

IBM System Application Architecture: Common User Access, Panel Design and User Interaction, IBM Document SC26–4351–0, Boca Raton, FL (December 1987), 328 pages.

—This older version of IBM's standards took years to prepare. It has been highly influential in the development of all IBM products, and therefore also of many corporate standards.

Microsoft, *The Windows Interface Guidelines for Software Design*, Microsoft Press, Redmond, WA (1995), 556 pages.

—This thoughtful analysis of usability principles (user in control, directness, consistency, forgiveness, aesthetics, and simplicity) gives detailed guidance for Windows software developers regarding how to make it happen.

Open Software Foundation, *OSF/Motif Style Guide* and *OSF/Motif User's Guide*, Prentice-Hall, Englewood Cliffs, NJ (1990).

—This book provides readable explanations for designers and for users to create or use applications under the OSF/Motif environment. Covers menus, windows, dialog boxes, and help facilities.

Books

Classic books

Bolt, Richard A., *The Human Interface: Where People and Computers Meet*, Lifelong Learning Publications, Belmont, CA (1984), 113 pages.

Cakir, A., Hart, D. J., and Stewart, T. F. M., *Visual Display Terminals: A Manual Covering Ergonomics, Workplace Design, Health and Safety, Task Organization*, John Wiley and Sons, New York (1980).

Card, Stuart K., Moran, Thomas P., and Newell, Allen, *The Psychology of Human–Computer Interaction*, Lawrence Erlbaum Associates, Hillsdale, NJ (1983), 469 pages.

Coats, R. B. and Vlaeminke, I., *Man–Computer Interfaces: An Introduction to Software Design and Implementation,* Blackwell Scientific Publications, Oxford, U.K. (1987), 381 pages.

Crawford, Chris, *The Art of Computer Game Design: Reflections of a Master Game Designer,* Osborne/McGraw-Hill, Berkeley, CA (1984), 113 pages.

Dreyfus, W., *The Measure of Man: Human Factors in Design* (Second Edition), Whitney Library of Design, New York (1967).

Dumas, Joseph S., *Designing User Interfaces for Software,* Prentice-Hall, Englewood Cliffs, NJ (1988), 174 pages.

Ehrich, R. W. and Williges, R. C., *Human–Computer Dialogue Design,* Elsevier Science Publishers B.V., Amsterdam, The Netherlands (1986).

Galitz, Wilbert O., *Human Factors in Office Automation,* Life Office Management Association, Atlanta, GA (1980), 237 pages.

Galitz, Wilbert O., *Handbook of Screen Format Design* (Third Edition), Q. E. D. Information Sciences, Wellesley, MA (1989), 307 pages.

Gilmore, Walter E., Gertman, David I., and Blackman, Harold S., *User–Computer Interface in Process Control: A Human Factors Engineering Handbook,* Academic Press, San Diego, CA (1989) 436 pages.

Hiltz, Starr Roxanne, *Online Communities: A Case Study of the Office of the Future,* Ablex, Norwood, NJ (1984), 261 pages.

Hiltz, Starr Roxanne and Turoff, Murray, *The Network Nation: Human Communication via Computer,* Addison-Wesley, Reading, MA (1978).

Kantowitz, Barry H. and Sorkin, Robert D., *Human Factors: Understanding People-System Relationships,* John Wiley and Sons, New York (1983), 699 pages.

Kearsley, Greg, *Online Help Systems: Design and Implementation,* Ablex, Norwood, NJ (1988), 115 pages.

Martin, James, *Design of Man–Computer Dialogues,* Prentice-Hall, Englewood Cliffs, NJ (1973), 509 pages.

Mehlmann, Marilyn, *When People Use Computers: An Approach to Developing an Interface,* Prentice-Hall, Englewood Cliffs, NJ (1981).

Mumford, Enid, *Designing Human Systems for New Technology,* Manchester Business School, Manchester, U.K. (1983), 108 pages.

National Research Council, Committee on Human Factors, *Research Needs for Human Factors,* National Academy Press, Washington, D.C. (1983), 160 pages.

Nickerson, Raymond S., *Using Computers: Human Factors in Information Systems,* MIT Press, Cambridge, MA (1986), 434 pages.

Norman, Donald A., *The Psychology of Everyday Things,* Basic Books, New York (1988), 257 pages.

Oborne, David J., *Computers at Work: A Behavioural Approach,* John Wiley and Sons, Chichester, U.K. (1985), 420 pages.

Roebuck, J. A., Kroemer, K. H. E., and Thomson, W. G., *Engineering Anthropometry Methods,* Wiley, New York (1975).

Rubinstein, Richard and Hersh, Harry, *The Human Factor: Designing Computer Systems for People,* Digital Press, Maynard, MA (1984), 249 pages.

Schiff, W., *Perception: An Applied Approach,* Houghton Mifflin, New York (1980).

Sheridan, T. B. and Ferrel, W. R., *Man–Machine Systems: Information, Control, and Decision Models of Human Performance,* MIT Press, Cambridge, MA (1974).

Shneiderman, Ben, *Software Psychology: Human Factors in Computer and Information Systems,* Little, Brown, Boston (1980), 320 pages.

Tichauer, E. R., *The Mechanical Basis of Ergonomics,* John Wiley and Sons, New York (1978).

Turkle, Sherry, *The Second Self: Computers and the Human Spirit,* Simon and Schuster, New York (1984).

Weinberg, Gerald M., *The Psychology of Computer Programming,* Van Nostrand Reinhold, New York (1971), 288 pages.

Weizenbaum, Joseph, *Computer Power and Human Reason: From Judgment to Calculation,* W. H. Freeman, San Francisco (1976), 300 pages.

Winograd, Terry and Flores, Fernando, *Understanding Computers and Cognition,* Ablex, Norwood, NJ (1986), 207 pages.

Zuboff, Shoshanna, *In the Age of the Smart Machine: The Future of Work and Power,* Basic Books, New York (1988), 468 pages.

Recent books

Bailey, Robert W., *Human Performance Engineering: Using Human Factors/Ergonomics to Achieve Computer Usability* (Third Edition), Prentice-Hall, Englewood Cliffs, NJ (1996), 636 pages.

Barfield, Lon, *The User Interface: Concepts & Design,* Addison-Wesley, Reading, MA (1993), 353 pages.

Bass, Len and Coutaz, Joelle, *Developing Software for the User Interface,* Addison-Wesley, Reading, MA (1991), 256 pages.

Brown, C. Marlin "Lin," *Human–Computer Interface Design Guidelines,* Ablex, Norwood, NJ (1988), 236 pages.

Brown, Judith R. and Cunningham, Steve, *Programming the User Interface: Principles and Examples,* John Wiley and Sons, New York (1989), 371 pages.

Carroll, John M., *The Nurnberg Funnel: Designing Minimalist Instruction for Practical Computer Skill,* MIT Press, Cambridge, MA (1990), 340 pages.

Carroll, John, M., *Scenario-Based Design: Envisioning Work and Technology in System Development,* John Wiley and Sons, New York (1995), 406 pages.

Cooper, Alan, *About Face: The Essentials of User Interface Design,* IDG Books Worldwide, Foster City, CA (1995), 580 pages.

Dix, Alan, Finlay, Janet, Abowd, Gregory, and Beale, Russell, *Human–Computer Interaction,* Prentice Hall, New York (1993), 570 pages.

Druin, Allison and Solomon, Cynthia, *Designing Multimedia Environments for Children: Computers Creativity and Kids,* John Wiley and Sons, New York (1996), 263 pages.

Duffy, Thomas M., Palmer, James E., and Mehlenbacher, Brad, *Online Help: Design and Evaluation,* Ablex, Norwood, NJ (1993), 260 pages.

Dumas, Joseph S. and Redish, Janice C., *A Practical Guide to Usability Testing*, Ablex, Norwood, NJ (1993), 304 pages.

Eberts, Ray E., *User Interface Design*, Prentice Hall, Englewood Cliffs, NJ (1993), 649 pages.

Fernandes, Tony, *Global Interface Design: A Guide to Designing International User Interfaces*, Academic Press Professional, Boston, MA (1995), 191 pages.

Foley, James D., van Dam, Andries, Feiner, Steven K., and Hughes, John F., *Computer Graphics: Principles and Practice* (Second Edition), Addison-Wesley, Reading, MA (1990), 1174 pages.

Galitz, Wilbert O., *It's Time to Clean Your Windows: Designing GUIs that Work*, John Wiley and Sons, New York (1994), 477 pages.

Heckel, Paul, *The Elements of Friendly Software Design (The New Edition)*, SYBEX, San Francisco (1991), 319 pages.

Hix, Deborah, and Hartson, H. Rex, *Developing User Interfaces: Ensuring Usability Through Product and Process*, John Wiley and Sons, New York (1993), 381 pages.

Kantowitz, Barry H. *Experimental Psychology: Understanding Psychological Research* (Fifth Edition), West, Minneapolis/St. Paul, MN (1994).

Kobara, Shiz, *Visual Design with OSF/Motif*, Addison-Wesley, Reading, MA (1991), 260 pages.

Krueger, Myron, *Artificial Reality II*, Addison-Wesley, Reading, MA (1991), 304 pages.

Landauer, Thomas K., *The Trouble with Computers: Usefulness, Usability, and Productivity*, MIT Press, Cambridge, MA (1995), 425 pages.

Laurel, Brenda, *Computers as Theater*, Addison-Wesley, Reading, MA (1991), 211 pages.

Marchionini, Gary, *Information Seeking in Electronic Environments*, Cambridge University Press, Cambridge, U.K. (1995), 224 pages.

Marcus, Aaron, *Graphic Design for Electronic Documents and User Interfaces*, ACM Press, New York (1992), 266 pages.

Mayhew, Deborah J., *Principles and Guidelines in Software User Interface Design*, Prentice Hall, Englewood Cliffs, NJ (1992), 619 pages.

Mullet, Kevin and Sano, Darrell, *Designing Visual Interfaces: Communication Oriented Techniques*, Sunsoft Press, Englewood Cliffs, NJ (1995), 277 pages.

Myers, Brad, *Creating User Interfaces by Demonstration*, Academic Press, New York (1988), 320 pages.

Newman, William M. and Lamming, Michael G., *Interactive Systems Design*, Addison-Wesley, Reading, MA (1995), 468 pages.

Nielsen, Jakob, *Designing User Interfaces for International Use*, Elsevier Science Publishers, Amsterdam, The Netherlands (1990).

Nielsen, Jakob, *Multimedia and Hypertext: The Internet and Beyond*, Academic Press, Cambridge, MA (1995), 480 pages.

Nielsen, Jakob, *Usability Engineering*, Academic Press, Boston, MA (1993), 358 pages.

Norman, Kent, *The Psychology of Menu Selection: Designing Cognitive Control at the Human/Computer Interface*, Ablex, Norwood, NJ (1991), 350 pages.

Olsen, Jr., Dan R., *User Interface Management Systems: Models and Algorithms*, Morgan Kaufmann, San Mateo, CA (1991), 256 pages.

Preece, Jenny, *A Guide to Usability: Human Factors in Computing*, Addison-Wesley, Reading, MA (1993), 144 pages.

Preece, Jenny, Rogers, Yvonne, Sharp, Helen, Benyon, David, Holland, Simon, and Carey, Tom, *Human–Computer Interaction*, Addison-Wesley, Reading, MA (1994), 773 pages.

Ravden, Susannah and Johnson, Graham, *Evaluating Usability of Human–Computer Interfaces*, Halsted Press Division of John Wiley and Sons, New York (1989), 126 pages.

Sanders, M. S. and McCormick, Ernest J., *Human Factors in Engineering and Design* (Seventh Edition), McGraw-Hill, New York (1993).

Schuler, Douglas, *New Community Networks: Wired for Change*, ACM Press, New York, and Addison-Wesley, Reading, MA (1996), 528 pages.

Shneiderman, Ben and Kearsley, Greg, *Hypertext Hands-On! An Introduction to a New Way of Organizing and Accessing Information*, Addison-Wesley, Reading, MA (1989), 165 pages and two disks.

Thimbleby, Harold, *User Interface Design*, ACM Press, New York (1990), 470 pages.

Thorell, L. G. and Smith, W. J., *Using Computer Color Effectively*, Prentice-Hall, Englewood Cliffs, NJ (1990), 258 pages.

Tognazzini, Bruce, *Tog on Interface*, Addison-Wesley, Reading, MA (1992), 331 pages.

Travis, David, *Effective Color Displays: Theory and Practice*, Academic Press, Harcourt Brace Jovanovich, London, U.K. (1991), 301 pages.

Turkle, Sherry, *Life on the Screen: Identity in the Age of the Internet*, Simon and Schuster, New York (1995).

Vaske, Jerry and Grantham, Charles, *Socializing the Human–Computer Environment*, Ablex, Norwood, NJ (1990), 290 pages.

Wickens, Christopher D., *Engineering Psychology and Human Performance: Second Edition*, HarperCollins, New York (1992), 560 pages.

Documentation

Brockmann, R. John, *Writing Better Computer User Documentation: From Paper to Hypertext: Version 2.0*, John Wiley and Sons, New York (1990), 365 pages.

Haramundanis, Katherine, *The Art of Technical Documentation*, Digital Press, Maynard, MA (1992), 267 pages.

Horton, William K., *Designing and Writing Online Documentation: Help Files to Hypertext*, John Wiley and Sons, New York (1990), 372 pages.

Price, Jonathan, *How to Write a Computer Manual*, Benjamin/Cummings, Menlo Park, CA (1984), 295 pages.

Weiss, Edmond H., *How to Write a Usable User Manual*, ISI Press, Philadelphia, PA (1985), 197 pages.

Reference resource

ACM, *Resources in Human–Computer Interaction*, ACM Press, New York (1990), 1197 pages.

Collections

Proceedings Human Factors in Computer Systems, Washington, D.C., ACM (March 15–17, 1982), 399 pages.

The following volumes are available from ACM Order Dept., P. O. Box 64145, Baltimore, MD 21264, or from Addison-Wesley Publishing Co., One Jacob Way, Reading, MA 01867.

Proceedings ACM CHI '83 Conference: Human Factors in Computing Systems, Ann Janda (Editor), Boston, MA (December 12–15, 1983).

Proceedings ACM CHI '85 Conference: Human Factors in Computing Systems, Lorraine Borman and Bill Curtis (Editors), San Francisco (April 14–18, 1985).

Proceedings ACM CHI '86 Conference: Human Factors in Computing Systems, Marilyn Mantei and Peter Orbeton (Editors), Boston, MA (April 13–17, 1986).

Proceedings ACM CHI + GI '87 Conference: Human Factors in Computing Systems, John M. Carroll and Peter P. Tanner (Editors), Toronto, Canada (April 5–9, 1987).

Proceedings ACM CHI '88 Conference: Human Factors in Computing Systems, Elliot Soloway, Douglas Frye, and Sylvia B. Sheppard (Editors), Washington, D.C. (May 15–19, 1988).

Proceedings ACM CHI '89 Conference: Human Factors in Computing Systems, Ken Bice and Clayton Lewis (Editors), Austin, TX (April 30–May 4, 1989).

Proceedings ACM CHI '90 Conference: Human Factors in Computing Systems, Jane Carrasco Chew and John Whiteside (Editors), Seattle, WA (April 1–5, 1990).

Proceedings ACM CHI '91 Conference: Human Factors in Computing Systems, Scott P. Robertson, Gary M. Olson, and Judith S. Olson (Editors), New Orleans, LA (April 27–May 2, 1991).

Proceedings ACM CHI '92 Conference: Human Factors in Computing Systems, Penny Bauersfeld, John Bennett, and Gene Lynch (Editors), Monterey, CA (May 3–7, 1992)

Proceedings ACM INTERCHI '93 Conference: Human Factors in Computing Systems, Stacey Ashlund, Kevin Mullet, Austin Henderson, Erik Hollnagel, and Ted White (Editors), Amsterdam, The Netherlands (April 24–29, 1993).

Proceedings ACM CHI '94 Conference: Human Factors in Computing Systems, Beth Adelson, Susan Dumais, and Judith Olson (Editors), Boston, MA (April 24–28, 1994).

Proceedings ACM CHI '95 Conference: Human Factors in Computing Systems, Irvin R. Katz, Robert Mack, and Linn Marks (Editors), Denver, CO (May 7–11, 1995).

Proceedings ACM CHI '96 Conference: Human Factors in Computing Systems, Michael J. Tauber, Victoria Bellotti, Robin Jeffries, Jock D. Mackinlay, and Jakob Nielsen (Editors), Vancouver, Canada (April 13–18, 1996).

Proceedings ACM CHI '97 Conference: Human Factors in Computing Systems, Steven Pemberton, Jennifer J. Preece, and Mary Beth Rosson (Editors), Atlanta, GA (March 22–27, 1997).

INTERACT '84: IFIP International Conference on Human–Computer Interaction, North-Holland, Amsterdam, The Netherlands (1984).

INTERACT '87: IFIP International Conference on Human–Computer Interaction, North-Holland, Amsterdam, The Netherlands (1987).

INTERACT '90: IFIP International Conference on Human–Computer Interaction, North-Holland, Amsterdam, The Netherlands (1990).

INTERACT '93: IFIP International Conference on Human–Computer Interaction, North-Holland, Amsterdam, The Netherlands (1993).

INTERACT '96: IFIP International Conference on Human–Computer Interaction, North-Holland, Amsterdam, The Netherlands (1996).

INTERACT '97: IFIP International Conference on Human–Computer Interaction, North-Holland, Amsterdam, The Netherlands (1996).

Classic collections

Badre, Albert and Shneiderman, Ben (Editors), *Directions in Human–Computer Interaction*, Ablex, Norwood, NJ (1980), 225 pages.

Blaser, A. and Zoeppritz, M. (Editors), *Enduser Systems and Their Human Factors*, Springer-Verlag, Berlin (1983), 138 pages.

Carey, Jane (Editor), *Human Factors in Management Information Systems*, Ablex, Norwood, NJ (1988), 289 pages.

Coombs, M. J. and Alty, J. L. (Editors), *Computing Skills and the User Interface*, Academic Press, New York (1981).

Carroll, John M. (Editor), *Interfacing Thought: Cognitive Aspects of Human–Computer Interaction*, MIT Press, Cambridge, MA (1987), 324 pages.

Curtis, Bill (Editor), *Tutorial: Human Factors in Software Development*, IEEE Computer Society, Los Angeles (1981), 641 pages.

Durrett, H. John (Editor), *Color and the Computer*, Academic Press (1987), 299 pages.

Guedj, R. A., Hagen, P. J. W., Hopgood, F. R. A., Tucker, H. A., and Duce, D. A. (Editors), *Methodology of Interaction*, North-Holland, Amsterdam, The Netherlands (1980), 408 pages.

Hartson, H. Rex (Editor), *Advances in Human–Computer Interaction*, Volume 1, Ablex, Norwood, NJ (1985), 290 pages.

Hartson, H. Rex and Hix, Deborah (Editors), *Advances in Human–Computer Interaction*, Volume 2, Ablex, Norwood, NJ (1988), 380 pages.

Helander, Martin (Editor), *Handbook of Human–Computer Interaction*, North-Holland, Amsterdam (1988), 1167 pages.

Hendler, James A. (Editor), *Expert Systems: The User Interface*, Ablex, Norwood, NJ (1987), 336 pages.

Klemmer, Edmund T. (Editor), *Ergonomics: Harness the Power of Human Factors in Your Business*, Ablex, Norwood, NJ (1989), 218 pages.

Larson, James A. (Editor), *Tutorial: End User Facilities in the 1980's*, IEEE Computer Society Press (EHO 198–2), New York (1982).

Monk, Andrew (Editor), *Fundamentals of Human–Computer Interaction*, Academic Press, London, U.K. (1984), 293 pages.

Muckler, Frederick A. (Editor), *Human Factors Review: 1984*, Human Factors Society, Santa Monica, CA (1984), 345 pages.

Nielsen, Jakob (Editor), *Coordinating User Interfaces for Consistency*, Academic Press, San Diego, CA (1989), 142 pages.

Norman, Donald A. and Draper, Stephen W. (Editors), *User Centered System Design: New Perspectives on Human–Computer Interaction*, Lawrence Erlbaum Associates, Hillsdale, NJ (1986).

Salvendy, Gavriel (Editor), *Human–Computer Interaction, Proceedings of the First USA–Japan Conference on Human–Computer Interaction*, Elsevier Science Publishers, Amsterdam, The Netherlands (1984), 470 pages.

Salvendy, Gavriel (Editor), *Handbook of Human Factors*, John Wiley and Sons, New York (1987), 1874 pages.

Salvendy, Gavriel (Editor), *Cognitive Engineering in the Design of Human–Computer Interaction and Expert Systems*, Elsevier, Amsterdam, The Netherlands (1987), 592 pages.

Salvendy, Gavriel, Sauter, Steven L., and Hurrell, Jr., Joseph J. (Editors), *Social, Ergonomic and Stress Aspects of Work with Computers*, Elsevier, Amsterdam, The Netherlands (1987), 373 pages.

Salvendy, Gavriel, Smith, Michael J. (Editors), *Designing and Using Human–Computer Interfaces and Knowledge Based Systems*, Elsevier, Amsterdam, The Netherlands (1989), 990 pages.

Shackel, Brian (Editor), *Man–Computer Interaction: Human Factors Aspects of Computers and People*. Sijthoff and Noordhoof Publishers, Amsterdam, The Netherlands (1981), 560 pages.

Sime, M. and Coombs, M. (Editors), *Designing for Human–Computer Communication*, Academic Press, New York (1983), 332 pages.

Smith, H. T. and Green, T. R. G. (Editors), *Human Interaction with Computers*, Academic Press, New York (1980).

Smith, Michael J. and Salvendy, Gavriel (Editors), *Work with Computers: Organizational, Management, Stress and Health Aspects*, Elsevier Science Publishers B.V., Amsterdam, The Netherlands (1989), 698 pages.

Thomas, John C. and Schneider, Michael L. (Editors), *Human Factors in Computer Systems*, Ablex, Norwood, NJ (1984), 276 pages.

Van Cott, H. P. and Kinkade, R. G. (Editors), *Human Engineering Guide to Equipment Design*, U.S. Superintendent of Documents, Washington, D.C. (1972), 752 pages.

Vassiliou, Yannis (Editor), *Human Factors and Interactive Computer Systems*, Ablex, Norwood, NJ (1984), 287 pages.

Sherr, Sol (Editor), *Input Devices*, Academic Press, San Diego, CA (1988), 301 pages.

Wiener, Earl L., and Nagel, David C. (Editors), *Human Factors in Aviation*, Academic Press, New York (1988), 684 pages.

Recent collections

Adler, Paul S. and Winograd, Terry (Editors), *Usability: Turning Technologies into Tools*, Oxford University Press, New York (1992), 208 pages.

Baecker, R., Grudin, J., Buxton, W., and Greenberg, S. (Editors), *Readings in Human–Computer Interaction: Towards the Year 2000*, Morgan Kaufmann, Los Altos, CA (1995), 950 pages.

Bias, Randolph, and Mayhew, Deborah (Editors), *Cost-Justifying Usability*, Academic Press, New York (1994).

Bullinger, H.-J. (Editor), *Human Aspects of Computing: Design and Use of Interactive Systems and Information Management*, Elsevier Science Publishers B.V., Amsterdam, The Netherlands (1991), 1367 pages.

Carey, Jane (Editor), *Human Factors in Information Systems: An Organizational Perspective*, Ablex, Norwood, NJ (1991), 376 pages.

Carey, Jane (Editor), *Human Factors in Information Systems: Emerging Theoretical Bases*, Ablex, Norwood, NJ (1995), 381 pages.

Carroll, John M. (Editor), *Designing Interaction: Psychology at the Human–Computer Interface*, Cambridge University Press, Cambridge, U.K. (1991), 333 pages.

Cockton, G., Draper, S. W., and Weir, G. R. S. (Editors), *People and Computers IX*, Cambridge University Press, Cambridge, U.K. (1994), 428 pages.

Greenberg, Saul (Editor), *Computer-Supported Cooperative Work and Groupware*, Academic Press, London, U.K. (1991), 423 pages.

Greenberg, Saul, Hayne, Stephen, and Rada, Roy (Editors), *Groupware for Real Time Drawing: A Designer's Guide*, McGraw-Hill, New York (1995).

Hartson, H. Rex and Hix, Deborah (Editors), *Advances in Human–Computer Interaction*, Volume 3, Ablex, Norwood, NJ (1992), 288 pages.

Hartson, H. Rex and Hix, Deborah (Editors), *Advances in Human–Computer Interaction*, Volume 4, Ablex, Norwood, NJ (1993), 292 pages.

Laurel, Brenda (Editor), *The Art of Human–Computer Interface Design*, Addison Wesley, Reading, MA (1990), 523 pages.

MacDonald, Lindsay and Vince, John (Editors), *Interacting with Virtual Environments*, John Wiley and Sons, New York (1994), 291 pages.

Myers, Brad A. (Editor), *Languages for User Interfaces*, Jones and Bartlett Publishers, Boston, MA (1992).

Nielsen, Jakob (Editor), *Advances in Human–Computer Interaction*, Volume 5, Ablex, Norwood, NJ (1993), 258 pages.

Perlman, Gary, Green, Georgia K., and Wogalter, Michael S., *Human Factors Perspectives on Human–Computer Interaction: Selections from Proceedings of Human Factors and Ergonomics Society Annual Meetings 1983–1994*, Santa Monica, CA (1995), 381 pages.

Rudisill, Marianne, Lewis, Clayton, Polson, Peter B., and McKay, Timothy D., *Human–Computer Interface Design: Success Stories, Emerging Methods and Real-World Context*, Morgan Kaufmann, San Francisco (1995), 408 pages.

Shackel, Brian and Richardson, Simon (Editors), *Human Factors for Informatics Usability*, Cambridge University Press, Cambridge, U.K. (1991), 438 pages.

Winograd, Terry (Editor), *Bringing Design to Software*, ACM Press, New York, and Addison-Wesley, Reading, MA (1996), 321 pages.

Videotapes

Video is an effective medium for presenting the dynamic, graphical, interactive nature of modern user interfaces. The Technical Video Program of the ACM SIGCHI conferences makes it possible to see excellent demonstrations of often-cited but seldom-seen systems.

All CHI videos can be ordered directly through ACM:

ACM Member Service Department, 1515 Broadway, New York, NY 10036. Email: acmhelp@acm.org Tel: (800) 342–6626 or (212) 626–0613. VHS NTSC and PAL versions are available (http://www.acm.org/sigchi/video):

Year	(Location)
CHI'97	(Atlanta, GA)
CHI'96	(Vancouver, CA)
CHI'95	(Denver, CO)

Older Issues (1994 and before) were published with ACM SIGGRAPH Video Review:

SVR Issue Number	Year	(Location)
97	CHI'94	(Boston)
88/89	CHI'93	(Amsterdam, Netherlands)
76/77	CHI'92	(Monterey, CA)
78/79	CHI'92	Special Videos and Future Scenarios
63/64/65	CHI'91	(New Orleans, LA)
55/56	CHI'90	(Seattle, WA)
57	CHI'90	All the Widgets (Special Instructional Issue)
45/46	CHI'89	(Austin, TX)
47/48	CHI'89	(Austin, TX)
58/59	CHI'88	(Washington, D.C.)
33/34	CHI+GI'87	(Toronto, Canada)
26/27	CHI'86	(Boston, MA)
18/19	CHI'85	(San Francisco, CA)
12/13	CHI'83	(Boston, MA)

User-Interface Strategies The University of Maryland Instructional Television produces a live satellite television program and sells the tapes. Telephone (301) 405–4905.

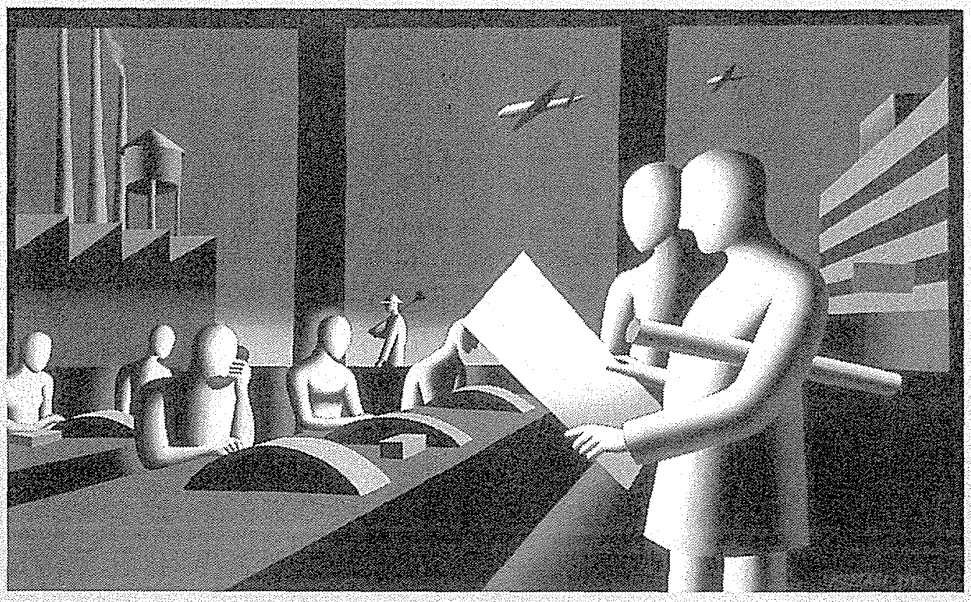
Email: itv@eng.umd.edu.http://www.glue.umd.edu/itv

The programs are coordinated by the author of this book who does at least a one-hour opening presentation followed by hour-long guest lectures and a discussion hour:

1996 Charles Kreitzberg and Edward Yourdon
1995 Frank Stein, Kent Norman, H. Rex Hartson, and Deborah Hix
1994 Jakob Nielsen, Judith Olson, and Myron Krueger
1993 Marilyn Mantei, Tom Furness, and James Martin
1992 Tom Landauer, Brad Myers, and Brenda Laurel
1991 Andries Van Dam, Elliot Soloway, and Bill Curtis
1990 Aaron Marcus, John Carroll, and Joy Mountford
1988 Tom Malone, Don Norman, and James Foley

Consulting and design companies

Aaron Marcus and Associates, Emeryville, CA
American Institutes for Research, Washington, D.C.
Cognetics Corp., Princeton Junction, NJ; Washington, D.C.
Dray & Associates, Minneapolis, MN
Ergo Research Group, Inc., Norwalk, CT
Human Factors International, Inc., Fairfield, IA
Preface User Interface Design, Burbank, CA
Usability Engineering Services, Inc., Kirkland, WA
Usernomics, Foster City, CA
UserWorks, Rockville, MD



Mark Kostabi, *The Art of Military Strategy*, 1997

CHAPTER

2

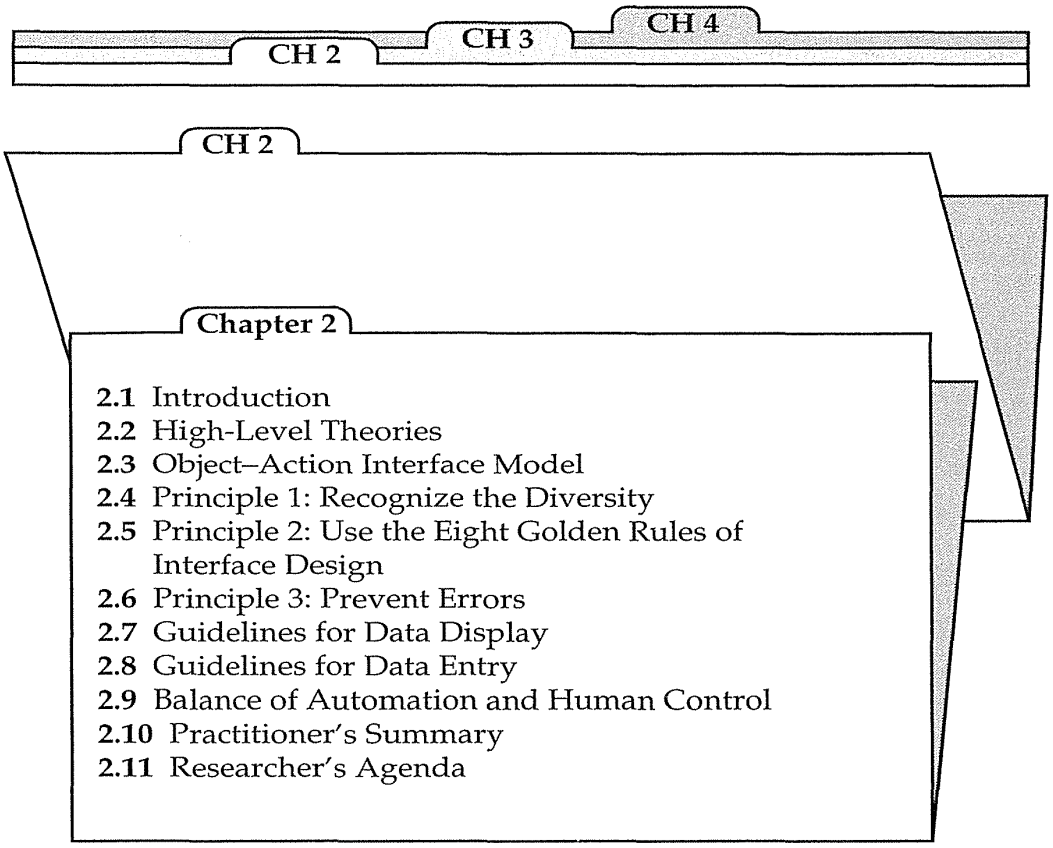
Theories, Principles, and Guidelines

We want principles, not only developed—the work of the closet—but applied, which is the work of life.

Horace Mann, ***Thoughts,*** **1867**

There never comes a point where a theory can be said to be true. The most that anyone can claim for any theory is that it has shared the successes of all its rivals and that it has passed at least one test which they have failed.

A.J. Ayer, ***Philosophy in the Twentieth Century,*** **1982**



2.1 Introduction

Successful designers of interactive systems know that they can and must go beyond intuitive judgments made hastily when a design problem emerges. Fortunately, guidance for designers is beginning to emerge in the form of (1) high-level theories and models, (2) middle-level principles, and (3) specific and practical guidelines. The theories and models offer a framework or language to discuss issues that are application independent, whereas the middle-level principles are useful in creating and comparing design alternatives. The practical guidelines provide helpful reminders of rules uncovered by designers.

In many contemporary systems, there is a grand opportunity to improve the user interface. The cluttered displays, complex and tedious procedures, inadequate functionality, inconsistent sequences of actions, and insufficient

informative feedback can generate debilitating stress and anxiety that lead to poor performance, frequent minor and occasional serious errors, and job dissatisfaction.

This chapter begins with a review of several theories, concentrating on the object–action interface model. Section 2.4 then deals with frequency of use, task profiles, and interaction styles. Eight golden rules of interface design are offered in Section 2.5. Strategies for preventing errors are described in Section 2.6. Specific guidelines for data entry and display appear in Sections 2.7 and 2.8. Sections 2.9 addresses the difficult question of balancing automation and human control.

2.2 High-Level Theories

Many theories are needed to describe the multiple aspects of interactive systems. Some theories are *explanatory*: They are helpful in observing behavior, describing activity, conceiving of designs, comparing high-level concepts of two designs, and training. Other theories are *predictive*: They enable designers to compare proposed designs for execution time or error rates. Some theories may focus on perceptual or cognitive subtasks (time to find an item on a display or time to plan the conversion of a boldfaced character to an italic one), whereas others concentrate on motor-task performance times. Motor-task predictions are the best established and are accurate for predicting keystroking or pointing times (see Fitts' Law, Section 9.3.5). Perceptual theories have been successful in predicting reading times for free text, lists, and formatted displays. Predicting performance on complex cognitive tasks (combinations of subtasks) is especially difficult because of the many strategies that might be employed and the many opportunities for going astray. The ratio for times to perform a complex task between novices and experts or between first-time and frequent users can be as high as 100 to 1. Actually, the contrast is even more dramatic because novices and first-time users often are unable to complete the tasks.

A *taxonomy* is a part of an explanatory theory. A taxonomy is the result of someone trying to put order on a complex set of phenomena; for example, a taxonomy might be created for input devices (direct versus indirect, linear versus rotary) (Card et al., 1990), for tasks (structured versus unstructured, controllable versus immutable) (Norman, 1991), for personality styles (convergent versus divergent, field dependent versus independent), for technical aptitudes (spatial visualization, reasoning) (Egan, 1988), for user experience levels (novice, knowledgeable, expert), or for user-interfaces styles (menus, form fillin, commands). Taxonomies facilitate useful comparisons, organize a topic for newcomers, guide designers, and often indicate opportunities for novel products.

Any theory that could help designers to predict performance for even a limited range of users, tasks, or designs would be a contribution (Card, 1989). For the moment, the field is filled with hundreds of theories competing for attention while being refined by their promoters, extended by critics, and applied by eager and hopeful—but skeptical—designers. This development is healthy for the emerging discipline of human–computer interaction, but it means that practitioners must keep up with the rapid developments, not only in software tools, but also in theories.

Another direction for theoreticians would be to try to predict subjective satisfaction or emotional reactions. Researchers in media and advertising have recognized the difficulty in predicting emotional reactions, so they complement theoretical predictions with their intuitive judgments and extensive market testing. Broader theories of small-group behavior, organizational dynamics, sociology of knowledge, and technology adoption may prove to be useful. Similarly, the methods of anthropology or social psychology may be helpful in understanding and overcoming barriers to new technology and resistance to change.

There may be "nothing so practical as a good theory," but coming up with an effective theory is often difficult. By definition, a theory, taxonomy, or model is an abstraction of reality and therefore must be incomplete. However, a good theory should at least be understandable, produce similar conclusions for all who use it, and help to solve specific practical problems.

2.2.1 Conceptual, semantic, syntactic, and lexical model

An appealing and easily comprehensible model is the four-level approach that Foley and van Dam developed in the late 1970s (Foley et al., 1990):

1. The *conceptual level* is the user's mental model of the interactive system. Two conceptual models for text editing are line editors and screen editors.
2. The *semantic level* describes the meanings conveyed by the user's command input and by the computer's output display.
3. The *syntactic level* defines how the units (words) that convey semantics are assembled into a complete sentence that instructs the computer to perform a certain task.
4. The *lexical level* deals with device dependencies and with the precise mechanisms by which a user specifies the syntax.

This approach is convenient for designers because its top-down nature is easy to explain, matches the software architecture, and allows for useful

modularity during design. Designers are expected to move from conceptual to lexical, and to record carefully the mappings between levels.

2.2.2 GOMS and the keystroke-level model

Card, Moran, and Newell (1980, 1983) proposed the *goals, operators, methods, and selection rules* (GOMS) model and the *keystroke-level model.* They postulated that users formulate goals (edit document) and subgoals (insert word), each of which they achieve by using methods or procedures (move cursor to desired location by following a sequence of arrow keys). The operators are "elementary perceptual, motor, or cognitive acts, whose execution is necessary to change any aspect of the user's mental state or to affect the task environment" (Card, et al. 1983, p. 144) (press up-arrow key, move hand to mouse, recall file name, verify that cursor is at end of file). The selection rules are the control structures for choosing among the several methods available for accomplishing a goal (delete by repeated backspace versus delete by placing markers at beginning and end of region and pressing delete button).

The keystroke-level model attempts to predict performance times for error-free expert performance of tasks by summing up the time for keystroking, pointing, homing, drawing, thinking, and waiting for the system to respond. These models concentrate on expert users and error-free performance, and place less emphasis on learning, problem solving, error handling, subjective satisfaction, and retention.

Kieras and Polson (1985) built on the GOMS approach and used production rules to describe the conditions and actions in an interactive text editor. The number and complexity of production rules gave accurate predictions of learning and performance times for five text-editing operations: insert, delete, copy, move, and transpose. Other strategies for modeling interactive-system usage involve *transition diagrams* (Fig. 2.1). These diagrams are helpful during design; for instruction; and as a predictor of learning time, performance time, and errors.

Kieras (1988), however, complains that the Card, Moran, and Newell presentation "does not explain in any detail how the notation works, and it seems somewhat clumsy to use. Furthermore, the notation has only a weak connection to the underlying cognitive theory." Kieras offers a refinement with his *Natural GOMS Language* (NGOMSL) and an analysis method for writing down GOMS models. He tries to clarify the situations in which the GOMS task analyst must make a *judgment call,* must make assumptions about how users view the system, must bypass a complex hard-to-analyze task (choosing wording of a sentence, finding a bug in a program), or must check for consistency. Applying NGOMSL to guide the process of creating online help, Elkerton and Palmiter (1991) developed *method descriptions* for their

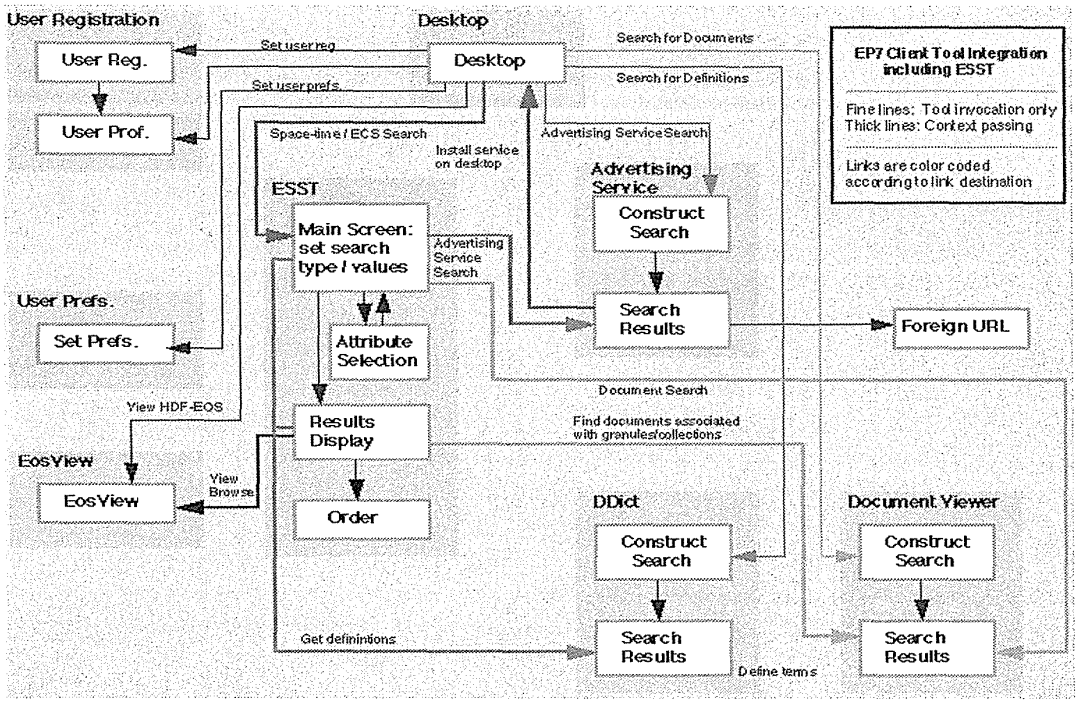


Figure 2.1

Transition diagram from the NASA search system.

interface, in which the actions necessary to accomplish a goal are broken down into steps. They also developed *selection rules*, by which a user can choose among alternative methods. For example, there may be two alternative methods to delete fields and one selection rule:

- Method 1 to accomplish the goal of deleting the field:
  - Step 1: Decide: If necessary, then accomplish the goal of selecting the field.
  - Step 2: Accomplish the goal of using a specific field-delete method.
  - Step 3: Report goal accomplished.
- Method 2 to accomplish the goal of deleting the field:
  - Step 1: Decide: If necessary, then use the Browse tool to go to the card with the field.
  - Step 2: Choose the Field tool in the Tools menu.
  - Step 3: Note that the fields on the card background are displayed.
  - Step 4: Click on the field to be selected.

Step 5: Report goal accomplished.

- Selection rule set for goal of using a specific field-delete method:
  - If you want to paste the field somewhere else, then choose "Cut Field" from the Edit menu.
  - If you want to delete the field permanently, then choose "Clear Field" from the Edit menu.
  - Report goal accomplished.

The empirical evaluation with 28 subjects demonstrated that the NGOMSL version of help halved the time users took to complete information searches in the first of four trial blocks.

A production-rule–based cognitive architecture called Soar provides a computer-based approach to implementing GOMS models. This software tool enables complex predictions of expert performance times based on perceptual and cognitive parameters. Soar was used to model learning in the highly interactive task of videogame playing (Bauer and John, 1995). John and Kieras (1996a, 1996b) compare four GOMS-related techniques and provide ten case studies of practical applications.

2.2.3 Stages of action models

Another approach to forming theories is to describe the stages of action that users go through in trying to use a system. Norman (1988) offers *seven stages of action* as a model of human–computer interaction:

1. Forming the goal
2. Forming the intention
3. Specifying the action
4. Executing the action
5. Perceiving the system state
6. Interpreting the system state
7. Evaluating the outcome

Some of Norman's stages correspond roughly to Foley and van Dam's separation of concerns; that is, the user forms a conceptual intention, reformulates it into the semantics of several commands, constructs the required syntax, and eventually produces the lexical token by the action of moving the mouse to select a point on the screen. Norman makes a contribution by placing his stages in the context of *cycles of action* and *evaluation*. This dynamic process of action distinguishes Norman's approach from the other models, which deal mainly with the knowledge that must be in the user's mind. Furthermore, the seven-stages model leads naturally to identification of the *gulf of execution* (the mismatch between the user's intentions and the

allowable actions) and the *gulf of evaluation* (the mismatch between the system's representation and the user's expectations).

This model leads Norman to suggest four principles of good design. First, the state and the action alternatives should be visible. Second, there should be a good conceptual model with a consistent system image. Third, the interface should include good mappings that reveal the relationships between stages. Fourth, the user should receive continuous feedback. Norman places a heavy emphasis on studying errors. He describes how errors often occur in moving from goals to intentions to actions and to executions.

A stages-of-action model helps us to describe user exploration of an interface (Polson and Lewis, 1990). As users try to accomplish their goals, there are four critical points where user failures can occur: (1) users can form an inadequate goal, (2) users might not find the correct interface object because of an incomprehensible label or icon, (3) users many not know how to specify or execute a desired action, and (4) users may receive inappropriate or misleading feedback. The latter three failures may be prevented by improved design or overcome by time-consuming experience with the interface (Franzke, 1995).

2.2.4 Consistency through grammars

An important goal for designers is a *consistent* user interface. However, the definition of consistency is elusive and has multiple levels that are sometimes in conflict; it is also sometimes advantageous to be inconsistent. The argument for consistency is that a command language or set of actions should be orderly, predictable, describable by a few rules, and therefore easy to learn and retain. These overlapping concepts are conveyed by an example that shows two kinds of inconsistency (A illustrates lack of any attempt at consistency, and B shows consistency except for a single violation):

Consistent	**Inconsistent A**	**Inconsistent B**
delete/insert character	delete/insert character	delete/insert character
delete/insert word	remove/bring word	remove/insert word
delete/insert line	destroy/create line	delete/insert line
delete/insert paragraph	kill/birth paragraph	delete/insert paragraph

Each of the actions in the consistent version is the same, whereas the actions vary for the inconsistent version A. The inconsistent action verbs are all acceptable, but their variety suggests that they will take longer to learn, will cause more errors, will slow down users, and will be harder for

users to remember. Inconsistent version B is somehow more malicious because there is a single unpredictable inconsistency that stands out so dramatically that this language is likely to be remembered for its peculiar inconsistency.

To capture these notions, Reisner (1981) proposed an *action grammar* to describe two versions of a graphics-system interface. She demonstrated that the version that had a simpler grammar was easier to learn. Payne and Green (1986) expanded her work by addressing the multiple levels of consistency (lexical, syntactic, and semantic) through a notational structure they call *task–action grammars* (TAGs). They also address some aspects of completeness of a language by trying to characterize a complete set of tasks; for example, *up*, *down*, and *left* constitute an incomplete set of arrow-cursor movement tasks, because *right* is missing. Once the full set of task–action mappings is written down, the grammar of the command language can be tested against it to demonstrate completeness. Of course, a designer might leave out something from the task–action mapping and then the grammar could not be checked accurately, but it does seem useful to have an approach to checking for completeness and consistency. For example, a TAG definition of cursor control would have a dictionary of tasks:

move-cursor-one-character-forward	[Direction = forward, Unit = char]
move-cursor-one-character-backward	[Direction = backward, Unit = char]
move-cursor-one-word-forward	[Direction = forward, Unit = word]
move-cursor-one-word-backward	[Direction = backward, Unit = word]

Then the high-level rule schemas that describe the syntax of the commands are as follows:

1. task [Direction, Unit] → symbol [Direction] + letter [Unit]
2. symbol [Direction = forward] → "CTRL"
3. symbol [Direction = backward] → "ESC"
4. letter [Unit = word] → "W"
5. letter [Unit = char] → "C"

These schemas will generate a consistent grammar:

move cursor one character forward	CTRL-C
move cursor one character backward	ESC-C
move cursor one word forward	CTRL-W
move cursor one word backward	ESC-W

Payne and Green are careful to state that their notation and approach are flexible and extensible, and they provide appealing examples in which their approach sharpened the thinking of designers.

Reisner (1990) extends this work by defining consistency more formally, but Grudin (1989) points out flaws in some arguments for consistency. Certainly consistency is subtle and has multiple levels; there are conflicting forms of consistency, and sometimes inconsistency is a virtue (for example, to draw attention to a dangerous operation). Nonetheless, understanding consistency is an important goal for designers and researchers.

2.2.5 Widget-level theories

Hierarchical decomposition is often a useful tool for dealing with complexity, but many of the theories and predictive models follow an extreme reductionist approach, which may not always be valid. In some situations, it is hard to accept the low level of detail, the precise numbers that are sometimes attached to subtasks, and the validity of simple summations of time periods. Furthermore, models requiring numerous subjective judgments raise the question of whether several analysts would come up with the same results.

An alternative approach is to follow the simplifications made in the higher-level, user-interface building tools (see Chapter 5). Instead of dealing with atomic level features, why not create a model based on the widgets (interface components) supported in the tool? Once a scrolling-list widget was tested to determine user performance as a function of the number of items and the size of the window, then future widget users would have automatic generation of performance prediction. The prediction would have to be derived from some declaration of the task frequencies, but the description of the interface would emerge from the process of designing the interface.

A measure of layout appropriateness (frequently used pairs of widgets should be adjacent, and the left-to-right sequence should be in harmony with the task-sequence description) would also be produced to guide the designer in a possible redesign. Estimates of the perceptual and cognitive complexity plus the motor load would be generated automatically (Sears, 1992). As widgets become more sophisticated and more widely used, the investment in determining the complexity of each widget will be amortized over the many designers and projects.

Gradually, higher-level patterns of usage are appearing, in much that way that Alexander describes has occurred in architecture (1977). Familiar pat-

terns of building fireplaces, stairways, or roofs become modular components that acquire names and are combined to form still larger patterns.

2.3 Object–Action Interface Model

Distinctions between syntax and semantics have long been made by compiler writers who sought to separate out the parsing of input text from the operations that were invoked by the text. A *syntactic–semantic model* of human behavior was originated to describe programming (Shneiderman, 1980) and was applied to database-manipulation facilities (Shneiderman, 1981), as well as to direct manipulation (Shneiderman, 1983). The early syntactic-semantic model made a major distinction between meaningfully acquired semantic concepts and rote-memorized syntactic details. Semantic concepts of the users's tasks were well-organized and stable in memory, whereas syntactic details of command languages were arbitrary and had to be rehearsed frequently to be maintained.

The maturing model described in this book's first edition stressed the separation between task-domain concepts (for example, stock-market portfolios) and the computer-domain concepts that represent them (for example, folders, spreadsheets, or databases). Then, this book's second edition amplified the important distinction between objects and actions. By now, the objects and actions have become the dominant features. In this third edition, the underlying theory of design will be called the *object–action interface* (OAI—let's pronounce it Oo-Ah!) *model.*

As GUIs have replaced command languages, intricate syntax has given way to relatively simple direct manipulations applied to visual representations of objects and actions. The emphasis is now on the visual display of user task objects and actions. For example, a collection of stock-market portfolios might be represented by leather folders with icons of engraved share certificates. Then, the actions are represented—by trashcans for deletion, or shelf icons to represent destinations for portfolio copying. Of course, there are syntactic aspects of direct manipulation, such as knowing whether to drag the file to the trashcan or to drag the trashcan to the folder, but the amount of syntax is small and can be thought of as being at the lowest level of the interface actions. Even syntactic forms such as double-clicking, mouse-down-and-wait, or gestures seem simple compared to the pages of grammars for early command languages.

Doing object–action design starts with understanding the task. That task includes the universe of real-world objects with which users work to accomplish their intentions and the actions that they apply to those objects. The

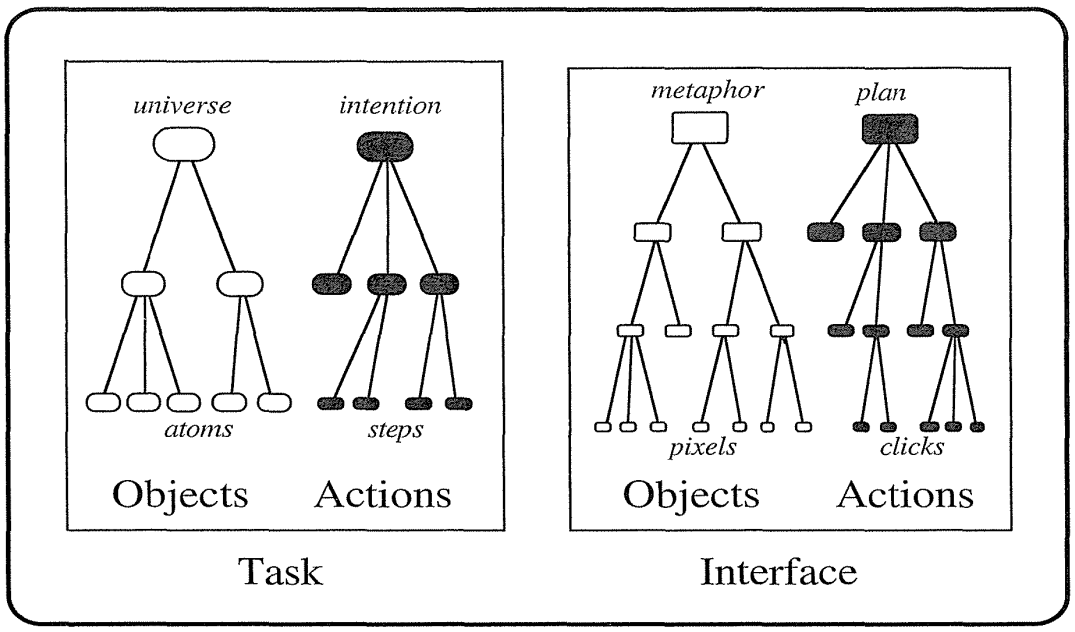


Figure 2.2

Task and interface concepts, separated into hierarchies of objects and actions.

high-level task objects might be stock-market statistics, a photo library, or a scientific journal (Fig. 2.2). These objects can be decomposed into information on a single stock and finally into atomic units such as a share price. Task actions start from high-level intentions that are decomposed into intermediate goals and individual steps.

Once there is agreement on the task objects and actions and their decomposition, the designer can create the metaphoric representations of the interface objects and actions. Interface objects do not have weight or thickness; they are pixels that can be moved or copied in ways that represent real-world task objects with feedback to guide users. Finally, the designer must make the interface actions visible to users, so that users can decompose their plan into a series of intermediate actions, such as opening a dialog box, all the way down to a series of detailed keystrokes and clicks.

In outline, the OAI model is an explanatory model that focuses on task objects and actions, and on interface objects and actions. Because the syntactic details are minimal, users who know the task domain objects and actions can learn the interface relatively easily (see Chapter 12). The OAI model also reflects the higher level of design with which most designers deal when they

use the widgets in user-interface–building tools. The standard widgets have familiar and simple syntax (click, double-click, drag, or drop) and simple forms of feedback (highlighting, scrolling, or movement), leaving the designer more focused on how to use these widgets to create a business-oriented solution. The OAI model is in harmony with the software-engineering trends toward object-oriented design and programming methods that have become popular in the past decade.

2.3.1 Task hierarchies of objects and actions

The primary way to deal with large and complex problems is to decompose them into several smaller problems in a hierarchical manner until each subproblem is manageable. For example, a human body is discussed in terms of neural, muscular, skeletal, reproductive, digestive, circulatory, and other subsystems, which in turn might be described by organs, tissues, and cells. Most real-world objects have similar decompositions: buildings, cities, computer programs, and plays, for example. Some objects are more neatly decomposed than are others; some objects are easier to understand than are others.

Similarly, intentions can be decomposed into smaller action steps. A building-construction plan can be reduced to a series of steps such as surveying the property, laying the foundation, building the frame, raising the roof, and completing the interior. A symphony performance has movements, measures, and notes; a baseball game has innings, outs, and pitches.

People learn the task objects and actions independently of their implementation on a computer. People learn about buildings or books through developmental experiences in their youth, but many tasks require specialized training, such as in how to manage stock-market portfolios, to design buildings, or to diagnose medical problems. It may take years to learn the terminology, to acquire the decision-making skills, and to become proficient.

Designers who develop computer systems to support professionals may have to take training courses, to read workbooks, and to interview users. Then, the designers can sit down and generate a hierarchy of objects and actions to model the users' tasks. This model forms a basis for designing the interface objects and actions plus their representation in pixels on a screen, in physical devices, or by a voice or other audio cue.

Users who must learn to use computers to accomplish real-world tasks must first become proficient in the task domain. An expert computer user who has not studied architecture will not be able to use a building-design package any more than a computer-savvy amateur can make reliable medical diagnoses.

In summary, tasks include hierarchies of objects and actions at high and low levels. Hierarchies are not perfect, but they are comprehensible and useful. Most users accept a separation of their tasks into high- and low-level objects and actions.

2.3.2 Interface hierarchies of objects and actions

The *interface* includes hierarchies of objects and actions at high and low levels. For example, a central set of *interface-object* concepts deals with storage. Users come to understand the high-level concept that computers store information. The stored information can be refined into objects, such as the directory and the files of information. In turn, the directory object is refined into a set of directory entries, each of which has a name, length, date of creation, owner, access control, and so on. Each file is an object that has a lower-level structure consisting of lines, fields, characters, fonts, pointers, binary numbers, and so on.

The *interface actions* also are decomposable into lower-level actions. The high-level plans, such as creating a text data file, may require load, insertion, and save actions. The midlevel action of saving a file is refined into the actions of storing a file and backup file on one of many disks, of applying access-control rights, of overwriting previous versions, of assigning a name to the file, and so on. Then, there are many low-level details about permissible file types or sizes, error conditions such as shortage of storage space, or responses to hardware or software errors. Finally, the low-level action of issuing a specific command is carried out by clicking on a pull-down menu item.

Designers craft interface objects and actions based on familiar examples, then tune those objects and actions to fit the task. For example, in developing a system to manage stock-market portfolios, the designer might consider spreadsheets, databases, word processors, or a specialized graphical design that allowed users to drag stock symbols to indicate buying or selling.

Users can learn interface objects and actions by seeing a demonstration, hearing an explanation of features, or conducting trial-and-error sessions. The metaphoric representation—abstract, concrete, or analogical—conveys the interface objects and actions. For example, to explain saving a file, an instructor might draw a picture of a disk drive and a directory to show where the file goes and how the directory references the file. Alternatively, the instructor might describe how the card catalog acts as a directory for books saved in the library.

When interface objects and actions have a logical structure that can be anchored to familiar task objects and actions, we expect that structure to be relatively stable in memory. If users remember the high-level concept of saving a file, they will be able to conclude that the file must have a name, a size,

and a storage location. The linkage to other objects and the visual presentation support the memorability of this knowledge.

These interface objects and actions were once novel, known by only a small number of scientists, engineers, and data-processing professionals. Now, these concepts are taught at the elementary-school level, argued over during coffee breaks in the office, and exchanged in the aisles of corporate jets. When educators talk of computer literacy, part of their plans cover these interface concepts.

The OAI model helps us to understand the multiple complex processes that must occur for users to be successful in using an interface to accomplish a task. For example, in writing a business letter using computer software, users have to integrate smoothly their knowledge of the task objects and actions and of the interface objects and actions. They must have the high-level concept of writing (task action) a letter (task object), recognize that the letter will be stored as a document (interface object), and know the details of the save command (interface action). Users must be fluent with the middle-level concept of composing a sentence, and must recognize the mechanisms for beginning, writing, and ending a sentence. Finally, users must know the proper low-level details of spelling each word (low-level task object), and must know where the keys are for each letter (low-level interface object). The goal of minimizing interface concepts (such as the syntax of a command language) while presenting a visual representation of the task objects and actions is the heart of the direct-manipulation approach to design (see Chapter 6).

Integrating the multiple levels of task and interface concepts is a substantial challenge that requires great motivation and concentration. Educational materials that facilitate the acquisition of this knowledge are difficult to design, especially because of the diversity of background knowledge and motivation levels of typical learners. The OAI model of user knowledge can provide a guide to educational designers by highlighting the different kinds of knowledge that users need to acquire (see Chapter 12) and a guide to web site designers (see Chapter 16).

Designers of interactive systems can apply the OAI model to systematize their work. Where possible, the task objects should be made explicit, and the user's task actions should be laid out clearly. Then, the interface objects and actions can be identified, and appropriate representations can be created. These designs are likely to increase comprehensibility to users and independence of specific hardware.

2.3.3 The disappearance of syntax

In the early days of computers, users had to maintain a profusion of device-dependent details in their human memories. These low-level syntactic details include the knowledge of which action erases a character (delete, backspace, CTRL-H, CTRL-G, CTRL-D, rightmost mouse button,

or ESCAPE), which action inserts a new line after the third line of a text file (CTRL-I, INSERT key, `I3`, `I 3`, or `3I`), which abbreviations are permissible, and which of the numbered function keys produces the previous screen.

The learning, use, and retention of this knowledge are hampered by two problems. First, these details vary across systems in an unpredictable manner. Second, acquiring syntactic knowledge is often a struggle because the arbitrariness of these minor design features greatly reduces the effectiveness of paired-associate learning. Rote memorization requires repeated rehearsals to reach competence, and retention over time is poor unless the knowledge is applied frequently. Syntactic knowledge is usually conveyed by example and repeated usage. Formal notations, such as Backus–Naur form, are useful for knowledgeable computer scientists, but are confusing to most users.

A further problem with syntactic knowledge, in some cases, lies in the difficulty of providing a hierarchical structure or even a modular structure to cope with the complexity. For example, how is a user to remember these details of using an electronic-mail system: press RETURN to terminate a paragraph, CTRL-D to terminate a letter, `Q` to quit the electronic-mail subsystem, and `logout` to terminate the session. The knowledgeable computer user understands these four forms of termination as commands in the context of the full system, but the novice may be confused by four seemingly similar situations that have radically different syntactic forms.

A final difficulty is that syntactic knowledge is system dependent. A user who switches from one machine to another may face different keyboard layouts, commands, function-key usage, and sequences of actions. Certainly there may be some overlap. For example, arithmetic expressions might be the same in two languages; unfortunately, however, the small differences can be the most annoying. One system uses `K` to keep a file and another uses `K` to kill the file, or `S` to save versus `S` to send.

Expert frequent users can overcome these difficulties, and they are less troubled by syntactic knowledge problems. Novices and knowledgeable users, however, are especially troubled by syntactic irregularities. Their burden can be lightened by use of menus (see Chapter 7), a reduction in the arbitrariness of the keypresses, use of consistent patterns of commands, meaningful command names and labels on keys, and fewer details that must be memorized (see Chapter 8).

Minimizing these burdens is the goal of most interface designers. Modern direct-manipulation styles (see Chapter 6) support the process of presenting users with screens filled with familiar objects and actions representing their task objects and actions. Modern user interface building tools (see Chapter 5) facilitate the design process by making standard widgets easily available.

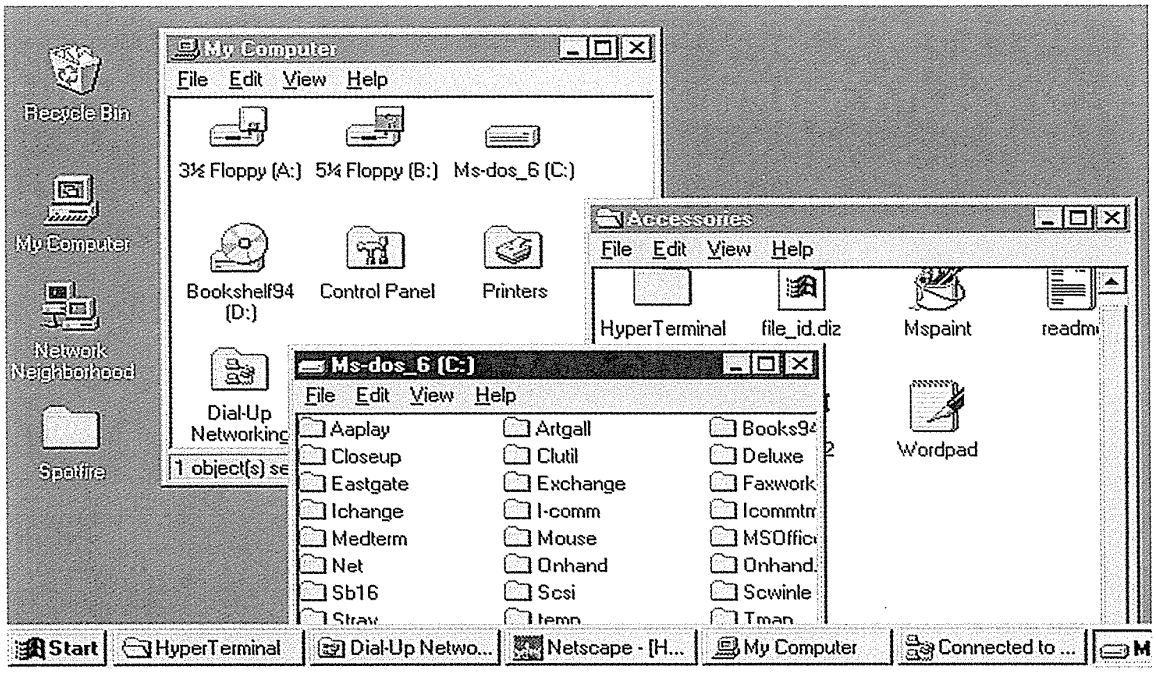


Plate A1: Microsoft Windows 95. (Reprinted with permission from Microsoft Corporation, Redmond, WA.)

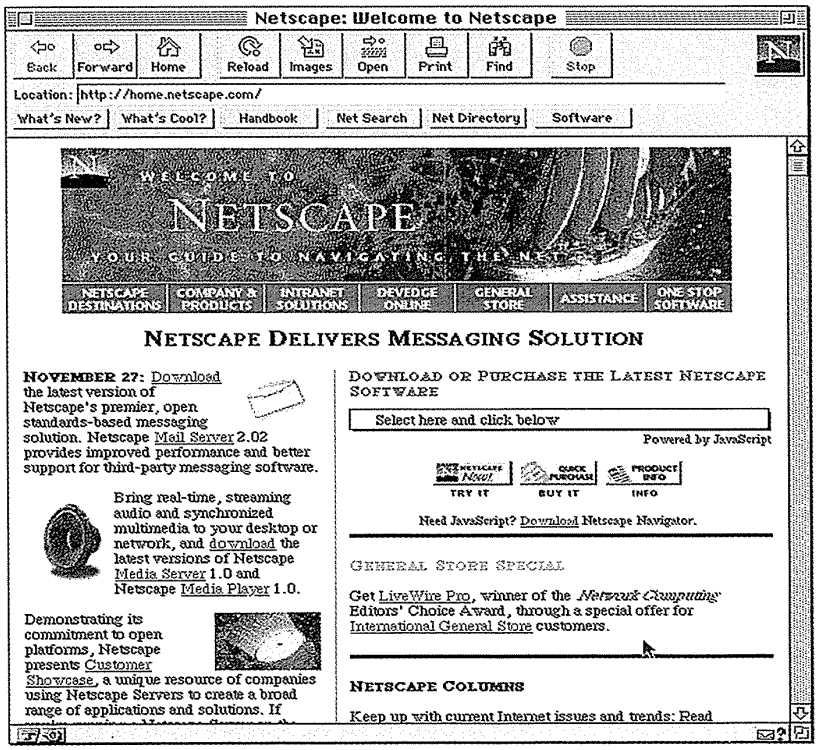


Plate A2: Netscape home page (http://home.netscape.com). (© 1996 Netscape Communication Corporation. Used with permission.)

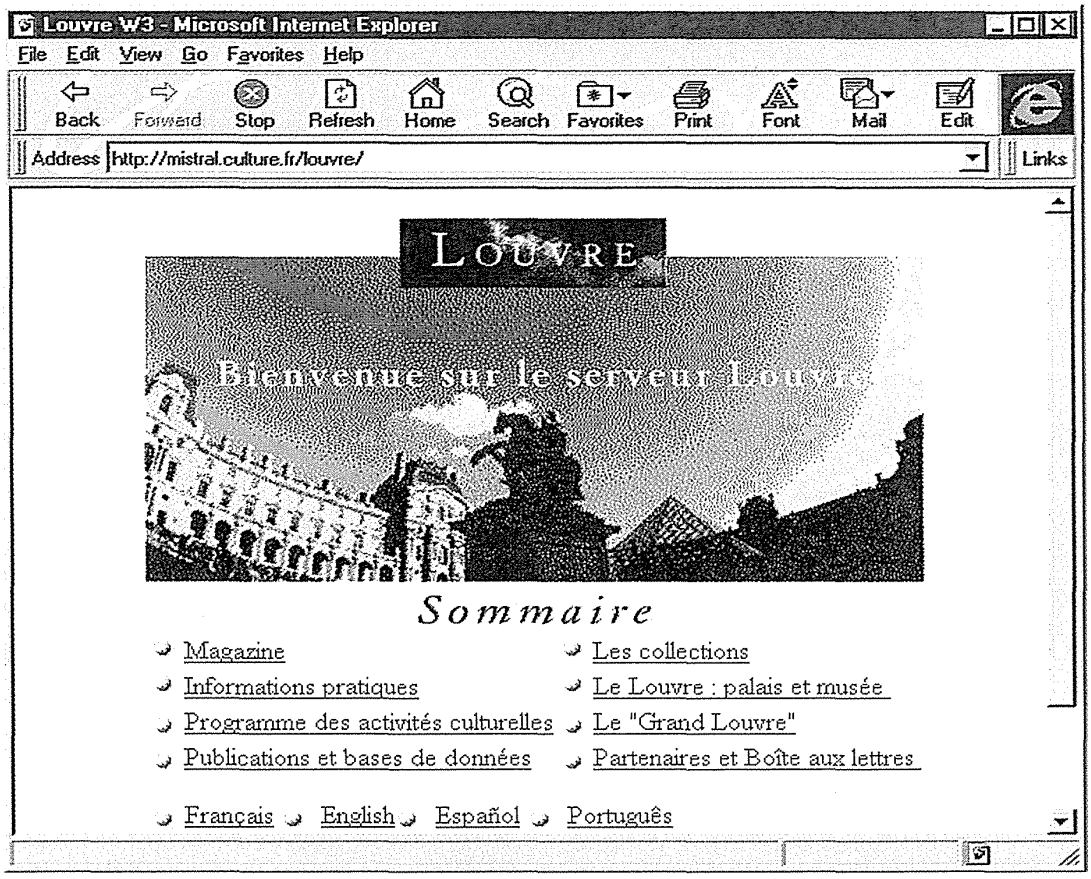


Plate A3: Microsoft Internet Explorer, showing the Louvre Museum web site (http://mistral.culture.fr/louvre).

Plate A4: White House home page (http://www.whitehouse.gov).

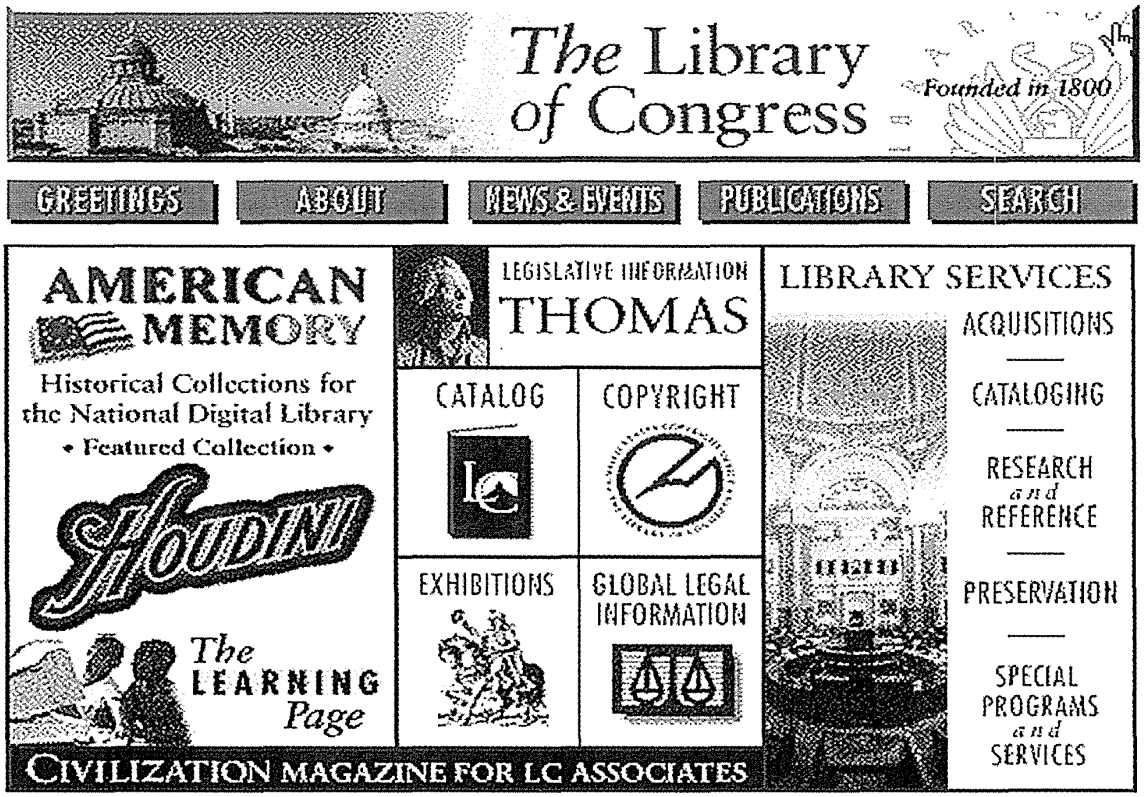


Plate A5: Library of Congress home page (http://www.loc.gov).

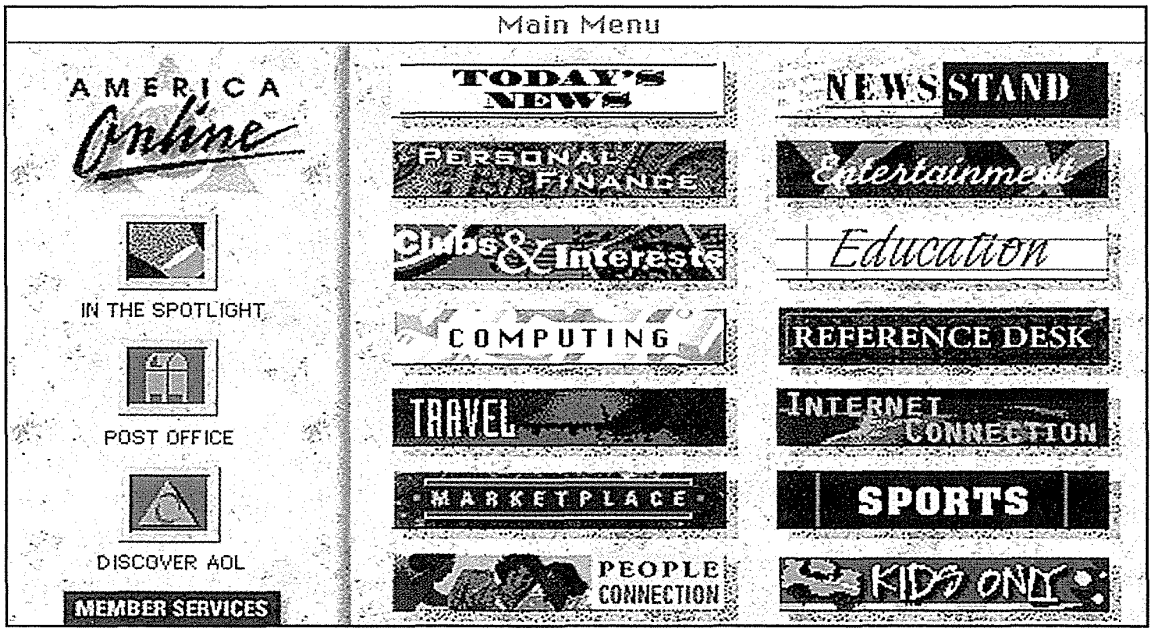


Plate A6: America Online main menu. (© 1997 America Online. Used by permission.)

Innovative designers may recognize opportunities for novel widgets that provide a closer match between the screen representation and the user's workplace.

2.4 Principle 1: Recognize the Diversity

When human diversity (see Section 1.5) is multiplied by the wide range of situations, tasks, and frequencies of use, the set of design possibilities becomes enormous. The designer can respond by choosing from a spectrum of interaction styles.

A preschooler playing a graphic computer game is a long way from a reference librarian doing bibliographic searches for anxious and hurried patrons. Similarly, a professional programmer using a new operating system is a long way from a highly trained and experienced air-traffic controller. Finally, a student surfing the net for love poems is a long way from a hotel-reservations clerk serving customers for many hours per day.

These sketches highlight the differences in users' background knowledge, training in the use of the system, frequency of use, and goals, as well as in the impact of a user error. No single design could satisfy all these users and situations, so before beginning a design, we must make the characterization of the users and the situation as precise and complete as possible.

2.4.1 Usage profiles

"Know thy user" was the first principle in Hansen's (1971) classic list of user-engineering principles. It is a simple idea, but a difficult and, unfortunately, often-undervalued goal. No one would argue against this principle, but many designers assume that they understand the users and users' tasks. Successful designers are aware that other people learn, think, and solve problems in different ways. Some users really do prefer to deal with tables rather than with graphs, with words instead of numbers, or with a rigid structure rather than an open-ended form.

It is difficult for most designers to know whether Boolean expressions are too difficult a concept for library patrons at a junior college, fourth graders learning programming, or professional controllers of electric-power utilities.

All design should begin with an understanding of the intended users, including population profiles that reflect age, gender, physical abilities, education, cultural or ethnic background, training, motivation, goals, and

personality. There are often several communities of users for a system, so the design effort is multiplied. Typical user communities—such as high school teachers, nurses, doctors, computer programmers, museum patrons, or librarians—can be expected to have various combinations of knowledge and usage patterns. Users from different countries may each deserve special attention, and even regional differences exist within countries. Other variables that characterize users include location (for example, urban vs. rural), economic profile, disabilities, and attitudes toward using technology.

In addition to these profiles, users might be tested for such skills as comprehension of Boolean expressions, knowledge of set theory, fluency in a foreign language, or skills in human relationships. Other tests might cover such task-specific abilities as knowledge of airport city codes, stockbrokerage terminology, insurance-claims concepts, or map icons.

The process of getting to know the users is never ending because there is so much to know and because the users keep changing. Every step in understanding the users and in recognizing them as individuals whose outlook is different from the designer's own is likely to be a step closer to a successful design.

For example, a generic separation into novice or first-time, knowledgeable intermittent, and expert frequent users might lead to these differing design goals:

- *Novice or first-time users* True novice users are assumed to know little of the task or interface concepts. By contrast, first-time users are professionals who know the task concepts, but have shallow knowledge of the interface concepts. Both groups of users may arrive with anxiety about using computers that inhibits learning. Overcoming these limitations is a serious challenge to the designer of the interface, including instructions, dialog boxes, and online help. Restricting vocabulary to a small number of familiar, consistently used concept terms is essential to begin developing the user's knowledge. The number of actions should also be small, so that novice and first-time users can carry out simple tasks successfully and thus reduce anxiety, build confidence, and gain positive reinforcement. Informative feedback about the accomplishment of each task is helpful, and constructive, specific error messages should be provided when users make mistakes. Carefully designed paper manuals and step-by-step online tutorials may be effective.
- *Knowledgeable intermittent users* Many people are knowledgeable but intermittent users of a variety of systems. They have stable task concepts and broad knowledge of interface concepts, but they will have difficulty retaining the structure of menus or the location of features. The burden on their memories will be lightened by orderly

structure in the menus, consistent terminology, and high interface apparency, which emphasizes recognition rather than recall. Consistent sequences of actions, meaningful messages, and guides to frequent patterns of usage will help knowledgeable intermittent users to rediscover how to perform their tasks properly. Protection from danger is necessary to support relaxed exploration of features or attempts to invoke a partially forgotten action sequence. These users will benefit from online help screens to fill in missing pieces of task or interface knowledge. Well-organized reference manuals also will be useful.

- *Expert frequent users* Expert "power" users are thoroughly familiar with the task and interface concepts and seek to get their work done quickly. They demand rapid response times, brief and nondistracting feedback, and the capacity to carry out actions with just a few keystrokes or selections. When a sequence of three or four commands is performed regularly, the frequent user is eager to create a *macro* or other abbreviated form to reduce the number of steps. Strings of commands, shortcuts through menus, abbreviations, and other accelerators are requirements.

These characteristics of these three classes of usage must be refined for each environment. Designing for one class is easy; designing for several is much more difficult.

When multiple usage classes must be accommodated in one system, the basic strategy is to permit a *level-structured* (some times called *layered* or *spiral approach*) to learning. Novices can be taught a minimal subset of objects and actions with which to get started. They are most likely to make correct choices when they have only a few options and are protected from making mistakes—when they are given a *training-wheels* interface. After gaining confidence from hands-on experience, these users can progress to ever-greater levels of task concepts and the accompanying interface concepts. The learning plan should be governed by the users' progress through the task concepts, with new interface concepts being introduced only when they are needed to support a more complex task. For users with strong knowledge of the task and interface concepts, rapid progress is possible.

For example, novice users of a bibliographic-search system might be taught author or title searches first, followed by subject searches that require Boolean combinations of queries. Their progress is governed by the task domain, rather than by an alphabetical list of commands that are difficult to relate to the tasks. The level-structured approach must be carried out in the design of not only the software, but also the user manuals, help screens, error messages, and tutorials.

Another approach to accommodating different usage classes is to permit user control of the density of informative feedback that the system provides. Novices want more informative feedback to confirm their actions, whereas frequent users want less distracting feedback. Similarly, it seems that frequent users like displays to be more densely packed than do novices. Finally, the pace of interaction may be varied from slow for novices to fast for frequent users.

2.4.2 Task profiles

After carefully drawing the user profile, the developers must identify the tasks. Task analysis has a long, but mixed, history (Bailey, 1996). Every designer would agree that the set of tasks must be determined before design can proceed, but too often the task analysis is done informally or implicitly. If implementers find that another command can be added, the designer is often tempted to include that command in the hope that some users will find it helpful. Design or implementation convenience should not dictate system functionality or command features.

High-level task actions can be decomposed into multiple middle-level task actions that can be further refined into atomic actions that the user executes with a single command, menu selection, and so on. Choosing the most appropriate set of atomic actions is a difficult task. If the atomic actions are too small, the users will become frustrated by the large number of actions necessary to accomplish a higher-level task. If the atomic actions are too large and elaborate, the users will need many such actions with special options, or they will not be able to get exactly what they want from the system.

The relative task frequencies will be important in shaping, for example, a set of commands or a menu tree. Frequently performed tasks should be simple and quick to carry out, even at the expense of lengthening some infrequent tasks. Relative frequency of use is one of the bases for making architectural design decisions. For example, in a text editor,

- Frequent actions might be performed by special keys, such as the four cursor arrows, INSERT, and DELETE.
- Intermediately frequent actions might be performed by a single letter plus CTRL, or by a selection from a pull-down menu—examples include underscore, center, indent, subscript, or superscript.
- Infrequent actions or complex actions might require going through a sequence of menu selections or form fillins—for example, to change the printing format or to revise network-protocol parameters.

A matrix of users and tasks can help us to sort out these issues (Fig. 2.3). In each box, the designer can put a check mark to indicate that this user carries

FREQUENCY OF TASK BY JOB TITLE

	Task				
Job title	*Query by Patient*	*Update Data*	*Query across Patients*	*Add Relations*	*Evaluate System*
Nurse	0.14	0.11			
Physician	0.06	0.04			
Supervisor	0.01	0.01	0.04		
Appointment personnel	0.26				
Medical-record maintainer	0.07	0.04	0.04	0.01	
Clinical researcher			0.08		
Database programmer			0.02	0.02	0.05

Figure 2.3

Hypothetical frequency-of-use data for a medical clinic information system. Answering queries from appointments personnel about individual patients is the highest-frequency task.

out this task. A more precise analysis would include frequencies instead of just simple check marks.

2.4.3 Interaction styles

When the task analysis is complete and the task objects and actions have been identified, the designer can choose from these primary interaction styles: menu selection, form fillin, command language, natural language, and direct manipulation (Box 2.1). Chapters 6 through 8 explore these styles in detail; here, we give a comparative overview to set the stage.

Direct manipulation When a clever designer can create a visual representation of the world of action, the users' tasks can be greatly simplified because direct manipulation of familiar objects is possible. Examples of such systems include the popular desktop metaphor, computer-assisted–design tools, air-traffic–control systems, and video games. By pointing at visual representations of objects and actions, users can carry out tasks rapidly and can observe the results immediately. Keyboard entry of commands or menu choices is

Box 2.1

Advantages and disadvantages of the five primary interaction styles.

Advantages	*Disadvantages*
Direct manipulation	
visually presents task concepts	may be hard to program
allows easy learning	may require graphics display and pointing devices
allows easy retention	
allows errors to be avoided	
encourages exploration	
affords high subjective satisfaction	
Menu selection	
shortens learning	presents danger of many menus
reduces keystrokes	may slow frequent users
structures decision making	consumes screen space
permits use of dialog-management tools	requires rapid display rate
allows easy support of error handling	
Form fillin	
simplifies data entry	consumes screen space
requires modest training	
gives convenient assistance	
permits use of form-management tools	
Command language	
is flexible	has poor error handling
appeals to "power" users	requires substantial training and memorization
supports user initiative	
allows convenient creation of user-defined macros	
Natural language	
relieves burden of learning syntax	requires clarification dialog
	may require more keystrokes
	may not show context
	is unpredictable

replaced by use of cursor-motion devices to select from a visible set of objects and actions. Direct manipulation is appealing to novices, is easy to remember for intermittent users, and, with careful design, it can be rapid for frequent users. Chapter 6 describes direct manipulation and its application.

Menu selection In menu-selection systems, users read a list of items, select the one most appropriate to their task, and observe the effect. If the terminology and meaning of the items are understandable and distinct, then users can accomplish their tasks with little learning or memorization and just a few actions. The greatest benefit may be that there is a clear structure to decision making, since all possible choices are presented at one time. This interaction style is appropriate for novice and intermittent users and can be appealing to frequent users if the display and selection mechanisms are rapid.

For designers, menu-selection systems require careful task analysis to ensure that all functions are supported conveniently and that terminology is chosen carefully and used consistently. Advanced user interface building tools to support menu selection are an enormous benefit in ensuring consistent screen design, validating completeness, and supporting maintenance.

Form fillin When data entry is required, menu selection usually becomes cumbersome, and form fillin (also called *fill in the blanks*) is appropriate. Users see a display of related fields, move a cursor among the fields, and enter data where desired. With the form-fillin interaction style, users must understand the field labels, know the permissible values and the data-entry method, and be capable of responding to error messages. Since knowledge of the keyboard, labels, and permissible fields is required, some training may be necessary. This interaction style is most appropriate for knowledgeable intermittent users or frequent users. Chapter 7 provides a thorough treatment of menus and form fillin.

Command language For frequent users, command languages provide a strong feeling of locus of control and initiative. Users learn the syntax and can often express complex possibilities rapidly, without having to read distracting prompts. However, error rates are typically high, training is necessary, and retention may be poor. Error messages and online assistance are hard to provide because of the diversity of possibilities plus the complexity of mapping from tasks to interface concepts and syntax. Command languages and lengthier query or programming languages are the domain of expert frequent users, who often derive great satisfaction from mastering a complex set of semantics and syntax.

Natural language The hope that computers will respond properly to arbitrary natural-language sentences or phrases engages many researchers and system

developers, in spite of limited success thus far. Natural-language interaction usually provides little context for issuing the next command, frequently requires *clarification dialog*, and may be slower and more cumbersome than the alternatives. Still, where users are knowledgeable about a task domain whose scope is limited and where intermittent use inhibits command-language training, there exist opportunities for natural-language interfaces (discussed at the end of Chapter 8).

Blending several interaction styles may be appropriate when the required tasks and users are diverse. Commands can lead the user to a form fillin where data entry is required, or menus can be used to control a direct-manipulation environment when a suitable visualization of actions cannot be found.

2.5 Principle 2: Use the Eight Golden Rules of Interface Design

Later chapters cover constructive guidance for design of direct manipulation, menu selection, command languages, and so on. This section presents underlying principles of design that are applicable in most interactive systems. These underlying principles of interface design, derived heuristically from experience, should be validated and refined.

1. *Strive for consistency.* This rule is the most frequently violated one, but following it can be tricky because there are many forms of consistency. Consistent sequences of actions should be required in similar situations; identical terminology should be used in prompts, menus, and help screens; and consistent color, layout, capitalization, fonts, and so on should be employed throughout. Exceptions, such as no echoing of passwords or confirmation of the delete command, should be comprehensible and limited in number.
2. *Enable frequent users to use shortcuts.* As the frequency of use increases, so do the user's desires to reduce the number of interactions and to increase the pace of interaction. Abbreviations, special keys, hidden commands, and macro facilities are appreciated by frequent knowledgeable users. Short response times and fast display rates are other attractions for frequent users.
3. *Offer informative feedback.* For every user action, there should be system feedback. For frequent and minor actions, the response can be modest, whereas for infrequent and major actions, the response should be more substantial. Visual presentation of the objects of interest provides a con-

venient environment for showing changes explicitly (see discussion of direct manipulation in Chapter 6).

4. *Design dialogs to yield closure.* Sequences of actions should be organized into groups with a beginning, middle, and end. The informative feedback at the completion of a group of actions gives operators the satisfaction of accomplishment, a sense of relief, the signal to drop contingency plans and options from their minds, and an indication that the way is clear to prepare for the next group of actions.
5. *Offer error prevention and simple error handling.* As much as possible, design the system such that users cannot make a serious error; for example, prefer menu selection to form fillin and do not allow alphabetic characters in numeric entry fields. If users make an error, the system should detect the error and offer simple, constructive, and specific instructions for recovery. For example, users should not have to retype an entire command, but rather should need to repair only the faulty part. Erroneous actions should leave the system state unchanged, or the system should give instructions about restoring the state.
6. *Permit easy reversal of actions.* As much as possible, actions should be reversible. This feature relieves anxiety, since the user knows that errors can be undone, thus encouraging exploration of unfamiliar options. The units of reversibility may be a single action, a data-entry task, or a complete group of actions such as entry of a name and address block.
7. *Support internal locus of control.* Experienced operators strongly desire the sense that they are in charge of the system and that the system responds to their actions. Surprising system actions, tedious sequences of data entries, inability or difficulty in obtaining necessary information, and inability to produce the action desired all build anxiety and dissatisfaction. Gaines (1981) captured part of this principle with his rule *avoid acausality* and his encouragement to make users the *initiators* of actions rather than the *responders* to actions.
8. *Reduce short-term memory load.* The limitation of human information processing in short-term memory (the rule of thumb is that humans can remember "seven-plus or minus-two chunks" of information) requires that displays be kept simple, multiple page displays be consolidated, window-motion frequency be reduced, and sufficient training time be allotted for codes, mnemonics, and sequences of actions. Where appropriate, online access to command-syntax forms, abbreviations, codes, and other information should be provided.

These underlying principles must be interpreted, refined, and extended for each environment. The principles presented in the ensuing sections

focus on increasing the productivity of users by providing simplified data-entry procedures, comprehensible displays, and rapid informative feedback that increase feelings of competence, mastery, and control over the system.

2.6 Principle 3: Prevent Errors

There is no medicine against death, and against error no rule has been found.

Sigmund Freud (Inscription he wrote on his portrait)

Users of word processors, spreadsheets, database-query facilities, air-traffic–control systems, and other interactive systems make mistakes far more frequently than might be expected. Card et al. (1980) reported that experienced professional users of text editors and operating systems made mistakes or used inefficient strategies in 31 percent of the tasks assigned to them. Brown and Gould (1987) found that even experienced authors made errors in almost half their spreadsheets. Other studies reveal the magnitude of the problem of—and the loss of productivity due to—user errors.

One way to reduce the loss in productivity due to errors is to improve the error messages provided by the computer system. Shneiderman (1982) reported on five experiments in which changes to error messages led to improved success at repairing the errors, lower error rates, and increased subjective satisfaction. Superior error messages were more specific, positive in tone, and constructive (telling the user what to do, rather than merely reporting the problem). Rather than using vague and hostile messages, such as `SYNTAX ERROR` or `ILLEGAL DATA`, designers were encouraged to use informative messages, such as `UNMATCHED LEFT PARENTHESIS` or `MENU CHOICES ARE IN THE RANGE OF 1 TO 6`.

Improved error messages, however, are only helpful medicine. A more effective approach is to prevent the errors from occurring. This goal is more attainable than it may seem in many systems.

The first step is to understand the nature of errors. One perspective is that people make mistakes or "slips" (Norman, 1983) that designers help them to avoid by organizing screens and menus functionally, designing commands or menu choices to be distinctive, and making it difficult for users to take irreversible actions. Norman offers other guidelines, such as do not have modes, do offer feedback about the state of the system, and do design for consistency of commands. Norman's analysis provides practical examples and a useful theory.

Three techniques can reduce errors by ensuring complete and correct actions: correct matching pairs, complete sequences, and correct commands.

2.6.1 Correct matching pairs

A common problem is the lack of correct matching pairs. It has many manifestations and several simple prevention strategies. An example is the failure to provide the right parenthesis to close an open left parenthesis. If a bibliographic-search system allowed Boolean expressions such as `COMPUTERS AND (PSYCHOLOGY OR SOCIOLOGY)` and the user failed to provide the right parenthesis at the end, the system would produce a `SYNTAX ERROR` message or, more helpfully, a more meaningful message, such as `UNMATCHED LEFT PARENTHESES`.

Similarly, other marker pairs are required to delimit boldface, italic, or underscored text in word processors or web programming. If the text file contains `<B>This is boldface</B>`, then the three words between the markers appear in boldface. If the rightmost `</B>` is missing, additional text may be inadvertently made bold.

In each of these cases, a matching pair of markers is necessary for operation to be complete and correct. The omission of the closing marker can be prevented by use of an editor, preferably screen oriented, that puts both the beginning and ending components of the pair on the screen in one action. For example, typing a left parenthesis generates a left and right parenthesis and puts the cursor in between to allow creation of the contents. An attempt to delete one of the parentheses will cause the matching parenthesis (and possibly the contents as well) to be deleted. Thus, the text can never be in a syntactically incorrect form. Some people find this rigid approach to be too restrictive. For them a milder form of protection may be appropriate. For example, when the user types a left parenthesis, the screen displays in the lower-left corner a message indicating the need for a right parenthesis until that character is typed.

2.6.2 Complete sequences

Sometimes, an action requires several steps or commands to reach completion. Since people may forget to complete every step of an action, designers attempt to offer a sequence of steps as a single action. In an automobile, the driver does not have to set two switches to signal a left turn. A single switch causes both (front and rear) turn-signal lights on the left side of the car to flash. When a pilot throws a switch to lower the landing gear, hundreds of steps and checks are invoked automatically.

This same concept can be applied to interactive uses of computers. For example, the sequence of dialing up, setting communication parameters, logging on,

and loading files is frequently executed by many users. Fortunately, most communications-software packages enable users to specify these processes once and then to execute them by simply selecting the appropriate name.

Users of a word processor should be able to indicate that section titles are to be centered, set in uppercase letters, and underlined, without having to issue a series of commands each time they enter a section title. Then, if the user wants to change the title style—for example, to eliminate underlining—a single command will guarantee that all section titles are revised consistently.

As a final example, air-traffic controllers may formulate plans to change the altitude of a plane from 14,000 feet to 18,000 feet in two increments; after raising the plane to 16,000 feet, however, the controller may get distracted and may thus fail to complete the action. The controller should be able to record the plan and then have the computer prompt for completion.

The notion of complete sequences of actions may be difficult to implement because users may need to issue atomic actions as well as complete sequences. In this case, users should be allowed to define sequences of their own; the macro or subroutine concept should be available at every level of usage.

Designers can gather information about potential complete sequences by studying sequences of commands that people actually issue, and the patterns of errors that people actually make.

2.6.3 Correct commands

Industrial designers recognize that successful products must be safe and must prevent the user from making dangerously incorrect use of the product. Airplane engines cannot be put into reverse until the landing gear has touched down, and cars cannot be put into reverse while traveling forward at faster than five miles per hour. Many simpler cameras prevent double exposures (even though the photographer may want to expose a frame twice), and appliances have interlocks to prevent tampering while the power is on (even though expert users occasionally need to perform diagnoses).

The same principles can be applied to interactive systems. Consider these typical errors made by the users of command languages: They invoke commands that are not available, request files that do not exist, or enter data values that are not acceptable. These errors are often caused by annoying typographic errors, such as using an incorrect command abbreviation; pressing a pair of keys, rather than a desired single key; misspelling a file name; or making a minor error such as omitting, inserting, or transposing characters. Error messages range from the annoyingly brief ?

or WHAT?, to the vague UNRECOGNIZED COMMAND or SYNTAX ERROR, to the condemning BAD FILE NAME or ILLEGAL COMMAND. The brief ? is suitable for expert users who have made a trivial error and can recognize it when they see the command line on the screen. But if an expert has ventured to use a new command and has misunderstood its operation, then the brief message is not helpful. They must interrupt their planning to deal with correcting the problem—and with their frustration in not getting what they wanted.

Some systems offer automatic command completion that allows users to type just a few letters of a meaningful command. They may request the computer to complete the command by pressing the space bar, or the computer may complete it as soon as the input is sufficient to distinguish the command from others. Automatic command completion can save keystrokes and is appreciated by many users, but it can also be disruptive because the user must consider how many characters to type for each command, and must verify that the computer has made the completion that was intended.

A more effective preventative for errors is to apply direct-manipulation strategies that emphasize selection over command-language typing. The computer presents permissible commands, menu choices, or file names on the screen, and users select their choice with a pointing device. This approach is effective if the screen has ample space, the display rate is rapid, and the pointing device is fast and accurate.

2.7 Guidelines for Data Display

The separation between basic principles and more informal guidelines is not a sharp line. However, thoughtful designers can distinguish between psychological principles (Wickens, 1993; Bridger, 1995) and practical guidelines that are gained from experience with a specific application. Guidelines for display of data are being developed by many organizations. A guidelines document can help by promoting consistency among multiple designers, recording practical experience, incorporating the results of empirical studies, and offering useful rules of thumb (see Chapters 3 and 11). The creation of a guidelines document engages the design community in a lively discussion of input or output formats, command sequences, terminology, and hardware devices (Brown, 1988; Galitz, 1993). Inspirations for design guidelines can also be taken from graphics designers (Tufte, 1983, 1990, 1997; Mullet and Sano, 1995).

2.7.1 Organizing the display

Smith and Mosier (1986) offer five high-level objectives for data display that remain vital:

1. *Consistency of data display* During the design process, the terminology, abbreviations, formats, colors, capitalization, and so on should all be standardized and controlled by use of a written (or computer-managed) dictionary of these items.
2. *Efficient information assimilation by the user* The format should be familiar to the operator and should be related to the tasks required to be performed with these data. This objective is served by rules for neat columns of data, left justification for alphanumeric data, right justification of integers, lining up of decimal points, proper spacing, use of comprehensible labels, and appropriate measurement units and numbers of decimal digits.
3. *Minimal memory load on user* Users should not be required to remember information from one screen for use on another screen. Tasks should be arranged such that completion occurs with few actions, minimizing the chance of forgetting to perform a step. Labels and common formats should be provided for novice or intermittent users.
4. *Compatibility of data display with data entry* The format of displayed information should be linked clearly to the format of the data entry. Where possible and appropriate, the output fields should also act as editable input fields.
5. *Flexibility for user control of data display* Users should be able to get the information from the display in the form most convenient for the task on which they are working. For example, the order of columns and sorting of rows should be easily changeable by users.

This compact set of high-level objectives is a useful starting point, but each project needs to expand these into application-specific and hardware-dependent standards and practices. For example, these generic guidelines emerge from a report on design of control rooms for electric-power utilities (Lockheed, 1981):

- Be consistent in labeling and graphic conventions.
- Standardize abbreviations.
- Use consistent format in all displays (headers, footers, paging, menus, and so on).
- Present a page number on each display page, and allow actions to call up a page via entry of a page number.
- Present data only if they assist the operator.

- Present information graphically where appropriate by using widths of lines, positions of markers on scales, and other techniques that relieve the need to read and interpret alphanumeric data.
- Present digital values only when knowledge of numerical value is necessary and useful.
- Use high-resolution monitors and maintain them to provide maximum display quality.
- Design a display in monochromatic form using spacing and arrangement for organization and then judiciously add color where it will aid the operator.
- Involve users in the development of new displays and procedures.

Chapter 11 further discusses data-display issues.

2.7.2 Getting the user's attention

Since substantial information may be presented to users for the normal performance of their work, exceptional conditions or time-dependent information must be presented so as to attract attention (Wickens, 1992). Multiple techniques exist for getting attention:

- *Intensity* Use two levels only, with limited use of high intensity to draw attention.
- *Marking* Underline, enclose in a box, point to with an arrow, or use an indicator such as an asterisk, bullet, dash, plus, or X.
- *Size* Use up to four sizes, with larger sizes attracting more attention.
- *Choice of fonts* Use up to three fonts.
- *Inverse video* Use inverse coloring.
- *Blinking* Use blinking displays (2 to 4 hertz) with great care and in limited areas.
- *Color* Use up to four standard colors, with additional colors reserved for occasional use.
- *Color blinking* Use changes in color (blinking from one color to another) with great care and in limited areas.
- *Audio* Use soft tones for regular positive feedback and harsh sounds for rare emergency conditions.

A few words of caution are necessary. There is a danger in creating cluttered displays by overusing these techniques. Novices need simple, logically organized, and well-labeled displays that guide their actions. Expert users do not

need extensive labels on fields; subtle highlighting or positional presentation is sufficient. Display formats must be tested with users for comprehensibility.

Similarly highlighted items will be perceived as being related. Color coding is especially powerful in linking related items, but this use makes it more difficult to cluster items across color codes. User control over highlighting—for example, allowing the operator in an air-traffic–control environment to assign orange to images of aircraft above 18,000 feet—may provide a useful resolution to concerns about personal preferences. Highlighting can be accomplished by increased intensity, blinking, or other methods.

Audio tones can provide informative feedback about progress, such as the clicks in keyboards or ringing sounds in telephones. Alarms for emergency conditions do alert users rapidly, but a mechanism to suppress alarms must be provided. If several types of alarms are used, testing is necessary to ensure that users can distinguish among alarm levels. Prerecorded or synthesized voice messages are an intriguing alternative, but since they may interfere with communications among operators, they should be used cautiously.

2.8 Guidelines for Data Entry

Data-entry tasks can occupy a substantial fraction of the operator's time and are the source of frustrating and potentially dangerous errors. Smith and Mosier (1986) offer five high-level objectives for data entry:

1. *Consistency of data-entry transactions* Similar sequences of actions should be used under all conditions; similar delimiters, abbreviations, and so on should be used.
2. *Minimal input actions by user* Fewer input actions mean greater operator productivity and—usually—fewer chances for error. Making a choice by a single keystroke, mouse selection, or finger press, rather than by typing in a lengthy string of characters, is potentially advantageous. Selecting from a list of choices eliminates the need for memorization, structures the decision-making task, and eliminates the possibility of typographic errors. However, if users must move their hands from a keyboard to a separate input device, the advantage is defeated because home-row position is lost. Experienced users often prefer to type six to eight characters instead of moving to a lightpen, joystick, or other selection device.

   A second aspect of this guideline is that redundant data entry should be avoided. It is annoying for users to enter the same information in two

locations, since the double entry is perceived as a waste of effort and an opportunity for error. When the same information is required in two places, the system should copy the information for the user, who still has the option of overriding by retyping.

3. *Minimal memory load on users* When doing data entry, users should not be required to remember lengthy lists of codes and complex syntactic command strings.
4. *Compatibility of data entry with data display* The format of data-entry information should be linked closely to the format of displayed information.
5. *Flexibility for user control of data entry* Experienced data-entry operators may prefer to enter information in a sequence that they can control. For example, on some occasions in an air-traffic control environment, the arrival time is the prime field in the controller's mind; on other occasions, the altitude is the prime field. Flexibility should be used cautiously, since it goes against the consistency principle.

2.9 Balance of Automation and Human Control

The principles described in the previous sections are in harmony with the goal of simplifying the user's task—eliminating human actions when no judgment is required. Users can then avoid the annoyance of handling routine, tedious, and error-prone tasks, and can concentrate on critical decisions, planning, and coping with unexpected situations (Sanders and McCormick, 1993). Computers should be used to keep track of and retrieve large volumes of data, to follow preset patterns, and to carry out complex mathematical or logical operations (Box 2.2 provides a detailed comparison of human and machine capabilities).

The degree of automation will increase over the years as procedures become more standardized, hardware reliability increases, and software verification and validation improves. With routine tasks, automation is preferred, since the potential for error may be reduced. However, I believe that there will always be a critical human role, because the real world is an *open system* (there is a nondenumerable number of unpredictable events and system failures). By contrast, computers constitute a *closed system* (there is only a denumerable number of normal and failure situations that can be accommodated in hardware and software). Human judgment is necessary for the unpredictable events in which some action must be taken to preserve safety, to avoid expensive failures, or to increase product quality (Hancock and Scallen, 1996).

Box 2.2

Relative capabilities of humans and machines. *Sources:* Compiled from Brown, 1988; Sanders and McCormick, 1993.

Humans Generally Better	Machines Generally Better
Sense low level stimuli	Sense stimuli outside human's range
Detect stimuli in noisy background	Count or measure physical quantities
Recognize constant patterns in varying situations	Store quantities of coded information accurately
Sense unusual and unexpected events	Monitor prespecified events, especially infrequent ones
Remember principles and strategies	Make rapid and consistent responses to input signals
Retrieve pertinent details without a priori connection	Recall quantities of detailed information accurately
Draw on experience and adapt decisions to situation	Process quantitative data in prespecified ways
Select alternatives if original approach fails	Reason deductively: infer from a general principle
Reason inductively: generalize from observations	Perform repetitive preprogrammed actions reliably
Act in unanticipated emergencies and novel situations	Exert great, highly-controlled physical force
Apply principles to solve varied problems	Perform several activities simultaneously
Make subjective evaluations	Maintain operations under heavy information load
Develop new solutions	Maintain performance over extended periods of time
Concentrate on important tasks when overload occurs	
Adapt physical response to changes in situation	

For example, in air-traffic control, common actions include changes to altitude, heading, or speed. These actions are well understood and can potentially be automatable by a scheduling and route-allocation algorithm, but the controllers must be present to deal with the highly variable and unpredictable emergency situations. An automated system might deal successfully with high volumes of traffic, but what would happen if the airport manager closed two runways because of turbulent weather? The controllers would have to reroute planes quickly. Now suppose that there is only one active runway and one pilot calls in to request special clearance to land because of a failed engine,

while another pilot in a second plane reports a passenger with a potential heart attack. Human judgment is necessary to decide which plane should land first, and how much costly and risky diversion of normal traffic is appropriate. Air-traffic controllers cannot just jump into the emergency; they must be intensely involved in the situation as it develops if they are to make an informed and rapid decision. In short, real-world situations are so complex that it is impossible to anticipate and program for every contingency; human judgment and values are necessary in the decision-making process.

Another example of the complexity of real-world situations in air-traffic control emerges from an incident on a Boeing 727 that had a fire on board near an airport. The controller cleared other traffic from the flight path and began to guide the plane in for a landing. The smoke was so thick that the pilot had trouble reading his instruments. Then the onboard transponder burned out, so the air-traffic controller could no longer read the plane's altitude from the situation display. In spite of these multiple failures, the controller and the pilot managed to bring down the plane quickly enough to save the lives of many—but not all—of the passengers. A computer could not have been programmed to deal with this particular unexpected series of events.

A tragic outcome of excess automation occurred during a 1995 flight to Cali, Colombia. The pilots relied on the automatic pilot and failed to realize that the plane was making a wide turn to return to a location that they had already passed. When the ground-collision alarm sounded, the pilots were too disoriented to pull up in time; they crashed 200 feet below the mountain peak.

The goal of system design in many applications is to give operators sufficient information about current status and activities, so that, when intervention is necessary, they have the knowledge and the capacity to perform correctly, even under partial failures. Increasingly, the human role is to respond to unanticipated situations, equipment failure, improper human performance, and incomplete or erroneous data (Eason, 1980; Sheridan, 1988; Billings, 1997).

The entire system must be designed and tested, not only for normal situations, but also for as wide a range of anomalous situations as can be anticipated. An extensive set of test conditions might be included as part of the requirements document. Operators need to have enough information that they can take responsibility for their actions.

Beyond performance of productive decision-making tasks and handling of failures, the role of the human operator will be to improve the design of the system. In complex systems, an opportunity always exists for improvement, so systems that lend themselves to refinement will evolve via continual incremental redesign by the operators.

The balance of automation and human control also emerges as an issue in systems for home and office automation. Some designers promote the notion of autonomous, adaptive, or anthropomorphic agents that carry out the users' intents and anticipate needs (Maes, 1994, 1995; Hayes-Roth, 1995; Hendler, 1996). Their scenarios often show a responsive, butler-like human

being to represent the agent (such as the bow-tied, helpful young man in Apple Computer's 1987 video on the *Knowledge Navigator*), or refer to the agent on a first-name basis (such as Sue or Bill in Hewlett-Packard's 1990 video on future computing). Microsoft's unsuccessful BOB program used cartoon characters to create onscreen partners. Other people have described *knowbots* or *softbots*—agents that traverse the World Wide Web in search of information of interest, such as where to find a low price for a Hawaiian tour.

Many people are attracted to the idea of a powerful functionary carrying out their tasks and watching out for their needs. The wish to create an autonomous agent that knows people's likes and dislikes, makes proper inferences, responds to novel situations, and performs competently with little guidance is strong for some designers. They believe that human–human interaction is a good model for human–computer interaction, and they seek to create computer-based partners, assistants, or agents. They promote their designs as intelligent and adaptive, and often they pursue anthropomorphic representations of the computer (see Section 11.3 for a review) to the point of having artificial faces talking to users. Anthropomorphic representations of computers have been unsuccessful in bank terminals, computer-assisted instruction, talking cars, and postal-service stations; however, these designers believe that they can find a way to attract users.

A variant of the agent scenario, which does not include an anthropomorphic realization, is that the computer employs a *user model* to guide an adaptive system. The system keeps track of user performance and adapts its behavior to suit the users' needs. For example, several proposals suggest that, as users make menu selections more rapidly, indicating proficiency, advanced menu items or a command-line interface appears. Automatic adaptations have been proposed for response time, length of messages, density of feedback, content of menus, order of menu items (see Section 7.3 for evidence against the helpfulness of this strategy), type of feedback (graphic or tabular), and content of help screens. Advocates point to video games that increase the speed or number of dangers as users progress though stages of the game. However, games are notably different from most work situations, where users have external goals and motivations to accomplish their tasks. There is much discussion of user models, but little empirical evidence of their efficacy.

There are some opportunities for adaptive user models to tailor system responses, but even occasional unexpected behavior has serious negative side effects that discourages use. If adaptive systems make surprising changes, users must pause to see what has happened. Then users may become anxious because they may not be able to predict the next change, interpret what has happened, or restore the system to the previous state. Suggestions that users could be consulted before a change is made are helpful, but such intrusions may still disrupt problem-solving processes and annoy users.

The agent metaphor is based on the design philosophy that assumes users would be attracted to "autonomous, adaptive, intelligent" systems. Designers

believe that they are creating a system that is lifelike and smart; however, users may feel anxious about and unable to control these systems. Success stories for advocates of adaptive systems include a few training and help systems that have been studied extensively and refined carefully to give users appropriate feedback for the errors that they make. Generalizing from these systems has proved to be more difficult than advocates had hoped.

These difficulties have led many agent proponents to shift to distributed World Wide Web searching and collaborative filtering (see Section 15.5). There is no visible agent or adaptation in the interface, but the applications aggregate information from multiple sources in some, often proprietary, way. Such blackbox approaches have great entertainment and even practical value in cases such as selecting movies, books, or music. However, in searching for antidotes in a toxicology database, physicians may want more predictable behavior and more control over what happens as they narrow their search.

The philosophical alternative to agents is *user-control, responsibility, and accomplishment*. Designers who emphasize a direct-manipulation style believe that users have a strong desire to be in control and to gain mastery over the system. Then, users can accept responsibility for their actions and derive feelings of accomplishment (Lanier, 1995; Shneiderman, 1995). Historical evidence suggests that users seek comprehensible and predictable systems and shy away from those that are complex or unpredictable; pilots may disengage automatic piloting devices if they perceive these systems are not performing as they expect.

Comprehensible and predictable user interfaces should mask the underlying computational complexity, in the same way that turning on an automobile ignition is comprehensible to users but invokes complex algorithms in the engine-control computer. These algorithms may adapt to varying engine temperatures or air pressures, but the action at the user-interface level remains predictable.

A critical issue for designers is the clear placement of responsibility for failures. Agent advocates usually avoid discussing responsibility, even for basic issues as violation of someone's copyright or for more serious flaws such as bugs that cause data destruction. Their designs rarely allow for monitoring the agent's performance, and feedback to users about the current user model is often given little attention. However, most human operators recognize and accept their responsibility for the operation of the computer, and therefore designers of financial, medical, or military applications ensure that detailed feedback is provided.

An alternative to agents and user models may be to expand the control-panel metaphor. Users use current control panels to set physical parameters, such as the speed of cursor blinking, rate of mouse tracking, or loudness of a speaker, and to establish personal preferences such as time and date formats, placement and format of menus, or color schemes (Figs. 2.4 and 2.5). Some software packages allow users to set parameters such as the speed of play in games or the usage level as in HyperCard (from browsing to editing buttons,

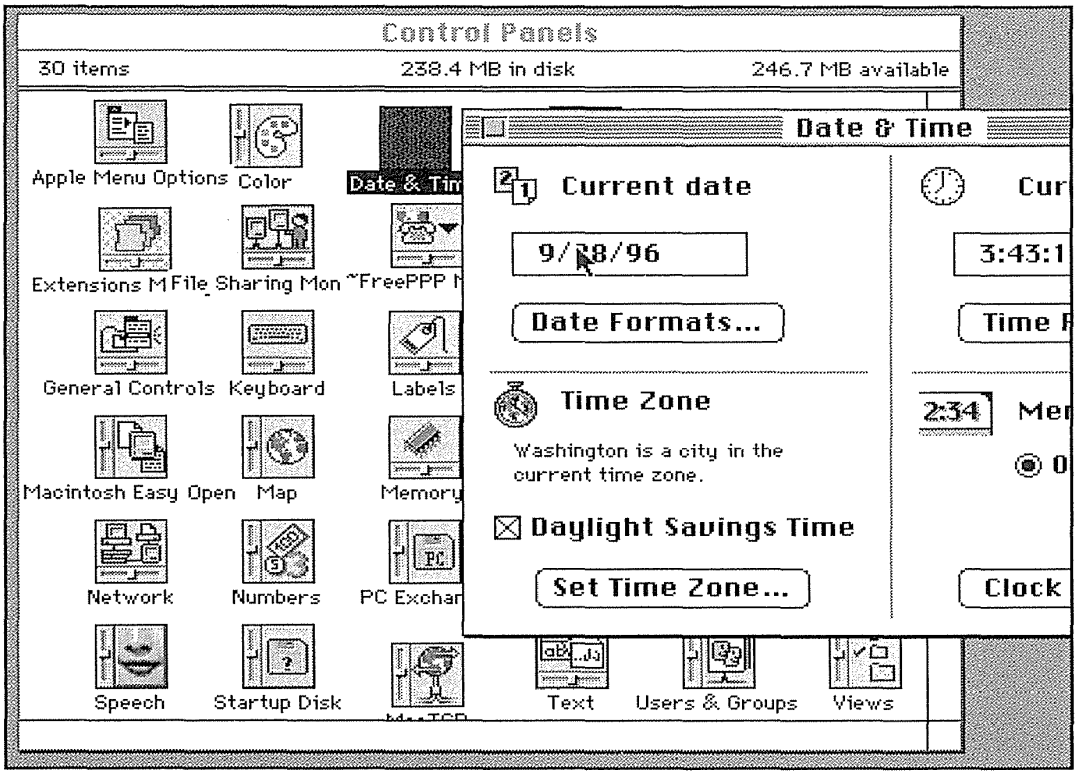


Figure 2.4

Macintosh MacOS 7.5 control panels, with `Date & Time` selected. Current control panels are used to set physical parameters (such as the speed of cursor blinking, rate of mouse tracking, or loudness of a speaker), and to establish personal preferences (such as time and date formats, placement and format of menus, or color schemes). (Used with permission of Apple Computer, Inc., Cupertino, CA.)

to writing scripts and creating graphics). Users start at level 1 and can then choose when to progress to higher levels. Often users are content remaining experts at level 1 of a complex system rather than dealing with the uncertainties of higher levels. More elaborate control panels exist in style sheets of word processors, specification boxes of query facilities, and information-visualization tools. Similarly, scheduling software may have elaborate controls to allow users to execute planned procedures at regular intervals or when triggered by other processes.

Computer control panels, like cruise-control mechanisms in automobiles and remote controllers for televisions, are designed to convey the sense of control that users seem to expect. Increasingly, complex processes are specified by direct-manipulation programming (see Chapter 6) or by graphical specifications of scheduled procedures, style sheets, and templates.

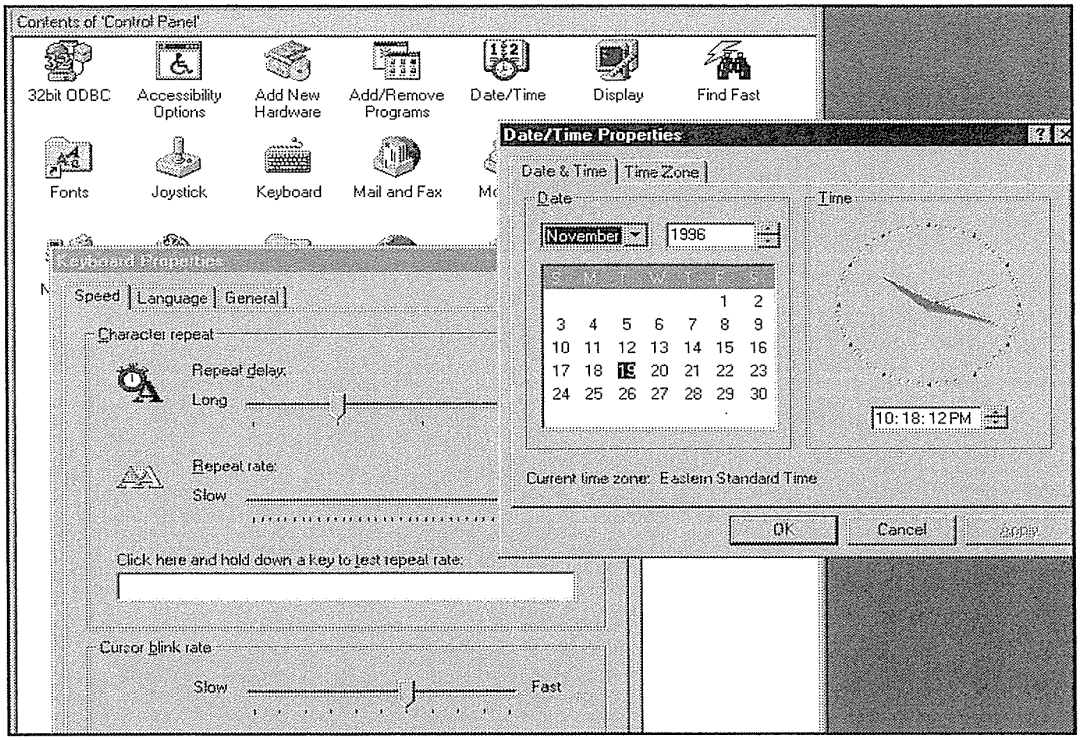


Figure 2.5

Microsoft Windows 95 control panel. (Used with permission of Microsoft Corp., Redmond, WA.)

2.10 Practitioner's Summary

Designing user interfaces is a complex and highly creative process that blends intuition, experience, and careful consideration of numerous technical issues. Designers are urged to begin with a thorough task analysis and a careful specification of the user communities. Explicit recording of task objects and actions can lead to construction of useful metaphors for interface objects and actions that benefit novice and expert users. Extensive testing and iterative refinement are necessary parts of every development project.

Design principles and guidelines are emerging from practical experience and empirical studies. Organizations can benefit by reviewing available guidelines documents and then constructing a local version. A guidelines document records organizational policies, supports consistency, aids the application of tools for user-interface building, facilitates training of new designers, records results of practice and experimental testing, and stimulates discussion of user-interface issues.

2.11 Researcher's Agenda

The central problem for psychologists, human-factors professionals, and computer scientists is to develop adequate theories and models of the behavior of humans who use interactive systems. Traditional psychological theories must be extended and refined to accommodate the complex human learning, memory, and problem-solving required in these applications. Useful goals include descriptive taxonomies, explanatory theories, and predictive models.

A first step might be to investigate thoroughly a limited task for a single community, and to develop a formal notation for describing task actions and objects. Then the mapping to interface actions and objects can be made precisely. This process would lead to predictions of learning times, performance speeds, error rates, subjective satisfaction, or human retention over time, for competing designs.

Next, the range of tasks and user communities could be expanded to domains of interest, such as word processing, information retrieval, or data entry. More limited and applied research problems are connected with each of the hundreds of design principles or guidelines that have been proposed. Each validation of these principles and clarification of the breadth of applicability would be a small but useful contribution to the emerging mosaic of human performance with interactive systems.

World Wide Web Resources WWW

Websites include theories and information on user models. A major topic with many websites is agents, including skeptical views. Debates over hot topics can be found in news groups which are searchable from many standard services such as Lycos or Infoseek.

http://www.aw.com/DTUI

References

Alexander, Christopher, Ishikawa, Sara, and Silverstein, Murray, *A Pattern Language: Towns, Buildings, Construction*, Oxford University Press, New York (1977).

Bailey, Robert W., *Human Performance Engineering: Using Human Factors/Ergonomics to Achieve Computer Usability* (Third Edition), Prentice-Hall, Englewood Cliffs, NJ (1996).

Bauer, Malcolm I., and John, Bonnie E., Modeling time-constrained learning in a highly interactive task, *Proc. CHI '95 Conference: Human Factors in Computing Systems*, ACM, New York (1996), 19–26

Billings, Charles E., *Animation Automation: The Search for a Human-Centered Approach*, Lawrence Erlbaum Assoc., Publishers, Mahwah, NJ (1997).

Bridger, R. S., *Introduction to Ergonomics*, McGraw-Hill, New York (1995).

Brown, C. Marlin, *Human–Computer Interface Design Guidelines*, Ablex, Norwood, NJ (1988).

Brown, P., and Gould, J., How people create spreadsheets, *ACM Transactions on Office Information Systems*, 5 (1987), 258–272.

Card, Stuart K., Theory-driven design research, in McMillan, Grant R., Beevis, David, Salas, Eduardo, Strub, Michael H., Sutton, Robert, and Van Breda, Leo (Editors), *Applications of Human Performance Models to System Design*, Plenum Press, New York (1989), 501–509.

Card, Stuart K., Mackinlay, Jock D., and Robertson, George G., The design space of input devices, *Proc. CHI '90 Conference: Human Factors in Computing Systems*, ACM, New York (1990), 117–124.

Card, Stuart, Moran, Thomas P., and Newell, Allen, The keystroke-level model for user performance with interactive systems, *Communications of the ACM*, 23 (1980), 396–410.

Card, Stuart, Moran, Thomas P., and Newell, Allen, *The Psychology of Human–Computer Interaction*, Lawrence Erlbaum Associates, Hillsdale, NJ (1983).

Eason, K. D., Dialogue design implications of task allocation between man and computer, *Ergonomics*, 23, 9 (1980), 881–891.

Eberts, Ray E., *User Interface Design*, Prentice Hall, Englewood Cliffs, NJ (1993).

Egan, Dennis E., Individual differences in human–computer interaction. In Helander, Martin (Editor), *Handbook of Human–Computer Interaction*, Elsevier Science Publishers, Amsterdam, The Netherlands (1988), 543–568.

Elkerton, Jay and Palmiter, Susan L., Designing help using a GOMS model: An information retrieval evaluation, *Human Factors*, 33, 2 (1991), 185–204.

Foley, James D., van Dam, Andries, Feiner, Steven K., and Hughes, John F., *Computer Graphics: Principles and Practice* (Second Edition), Addison-Wesley, Reading, MA (1990).

Franzke, Marita, Turning research into practice: Characteristics of display-based interaction, *Proc. CHI '95 Conference: Human Factors in Computing Systems*, ACM, New York (1995), 421–428.

Gaines, Brian R., The technology of interaction: Dialogue programming rules, *International Journal of Man–Machine Studies*, 14, (1981), 133–150.

Galitz, Wilbert O., *It's Time to Clean Your Windows: Designing GUIs that Work*, John Wiley and Sons, New York (1994).

Gilbert, Steven W., Information technology, intellectual property, and education, *EDUCOM Review*, 25, (1990), 14–20.

Grudin, Jonathan, The case against user interface consistency, *Communications of the ACM*, 32, 10 (1989), 1164–1173.

Hancock, P. A. and Scallen, S. F., The future of function allocation, *Ergonomics in Design*, 4, 4 (October 1996), 24–29.

Hansen, Wilfred J., User engineering principles for interactive systems, *Proc. Fall Joint Computer Conference*, 39, AFIPS Press, Montvale, NJ (1971), 523–532.

Hayes-Roth, Barbara, An architecture for adaptive intelligent systems, *Artificial Intelligence: Special Issue on Agents and Interactivity*, 72, (1995), 329–365.

Hendler, James A. (Editor), Intelligent agents: Where AI meets information technology, Special Issue, *IEEE Expert: Intelligent Systems & Their Applications 11*, 6 (December 1996), 20–63.

John, Bonnie and Kieras, David E., Using GOMS for user interface design and evaluation: Which technique? *ACM Transactions on Computer–Human Interaction* 3, 4 (December 1996a), 287–319.

John, Bonnie and Kieras, David E., The GOMS family of user interface analysis techniques: Comparison and contrast, *ACM Transactions on Computer–Human Interaction* 3, 4 (December 1996b), 320–351.

Kieras, David, Towards a practical GOMS model methodology for user interface design. In Helander, Martin (Editor), *Handbook of Human–Computer Interaction*, Elsevier Science Publishers, Amsterdam, The Netherlands (1988), 135–157.

Kieras, David, and Polson, Peter G., An approach to the formal analysis of user complexity, *International Journal of Man–Machine Studies*, 22, (1985), 365–394.

Lanier, Jaron, Agents of alienation, *ACM interactions*, 2, 3 (1995), 66–72

Lockheed Missiles and Space Company, *Human Factors Review of Electric Power Dispatch Control Centers. Volume 2: Detailed Survey Results*, (Prepared for) Electric Power Research Institute, Palo Alto, CA (1981).

Maes, Pattie, Agents that reduce work and information overload, *Communications of the ACM*, 37, 7 (July 1994), 31–40.

Maes, Pattie, Artificial life meets entertainment: Lifelike autonomous agents, *Communications of the ACM*, 38, 11 (November 1995), 108–114.

Mullet, Kevin and Sano, Darrell, *Designing Visual Interfaces: Communication Oriented Techniques*, Sunsoft Press, Englewood Cliffs, NJ (1995).

National Research Council, *Intellectual Property Issues in Software*, National Academy Press, Washington, D.C. (1991).

Norman, Donald A., Design rules based on analyses of human error, *Communications of the ACM*, 26, 4 (1983), 254–258.

Norman, Donald A., *The Psychology of Everyday Things*, Basic Books, New York (1988).

Norman, Kent L., Models of the mind and machine: Information flow and control between humans and computers, *Advances in Computers*, 32, (1991), 119–172.

Panko, Raymond R. and Halverson, Jr., Richard P., Spreadsheets on trial: A survey of research on spreadsheet risks, *Proc. Twenty-Ninth Hawaii International Conference on System Sciences* (1996).

Payne, S. J., and Green, T. R. G., Task-action grammars: A model of the mental representation of task languages, *Human–Computer Interaction*, 2, (1986), 93–133.

Payne, S. J., and Green, T. R. G., The structure of command languages: An experiment on task-action grammar, *International Journal of Man–Machine Studies*, 30, (1989), 213–234.

Polson, Peter, and Lewis, Clayton, Theory-based design for easily learned interfaces, *Human–Computer Interaction*, 5, (1990), 191–220.

Reisner, Phyllis, Formal grammar and design of an interactive system, *IEEE Transactions on Software Engineering*, SE–5, (1981), 229–240.

Reisner, Phyllis, What is consistency? In Diaper et al. (Editors), *INTERACT '90: Human–Computer Interaction*, North-Holland, Amsterdam, The Netherlands (1990), 175–181.

Sanders, M. S. and McCormick, Ernest J., *Human Factors in Engineering and Design* (Seventh Edition), McGraw-Hill, New York (1993).

Sears, Andrew, *Widget-Level Models of Human–Computer Interaction: Applying Simple Task Descriptions to Design and Evaluation*, PhD. Dissertation, Department of Computer Science, University of Maryland, College Park, MD (1992).

Sheridan, Thomas B., Task allocation and supervisory control. In Helander, M. (Editor), *Handbook of Human–Computer Interaction*, Elsevier Science Publishers, Amsterdam, The Netherlands (1988), 159–173.

Shneiderman, Ben, *Software Psychology: Human Factors in Computer and Information Systems*, Little, Brown, Boston, MA (1980).

Shneiderman, Ben, A note on the human factors issues of natural language interaction with database systems, *Information Systems*, 6, 2 (1981), 125–129.

Shneiderman, Ben, System message design: Guidelines and experimental results. In Badre, A. and Shneiderman, B. (Editors) *Directions in Human–Computer Interaction*, Ablex, Norwood, NJ (1982), 55–78.

Shneiderman, Ben, Direct manipulation: A step beyond programming languages, *IEEE Computer*, 16, 8 (1983), 57–69.

Shneiderman, Ben, Looking for the bright side of agents, *ACM Interactions*, 2, 1 (January 1995), 13–15.

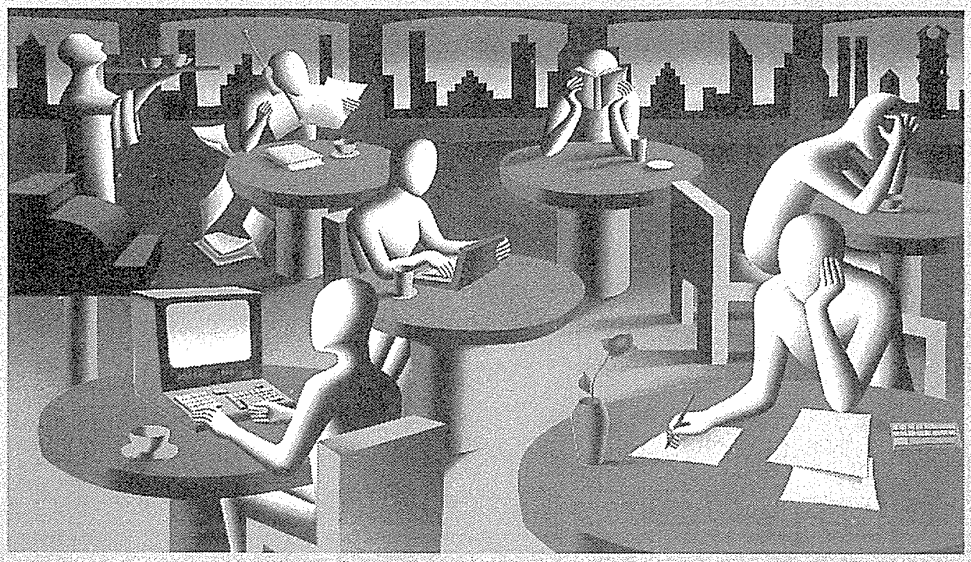
Smith, Sid L. and Mosier, Jane N., *Guidelines for Designing User Interface Software*, Report ESD-TR–86–278, Electronic Systems Division, MITRE Corporation, Bedford, MA (1986). Available from National Technical Information Service, Springfield, VA.

Tufte, Edward, *The Visual Display of Quantitative Information*, Graphics Press, Cheshire, CT (1983).

Tufte, Edward, *Envisioning Information*, Graphics Press, Cheshire, CT (1990).

Tufte, Edward, *Visual Explanations*, Graphics Press, Cheshire, CT (1997).

Wickens, Christopher D., *Engineering Psychology and Human Performance* (Second Edition), HarperCollins Publishers, New York (1992).



Mark Kostabi, *Computer Klatsch*, 1996

CHAPTER

3

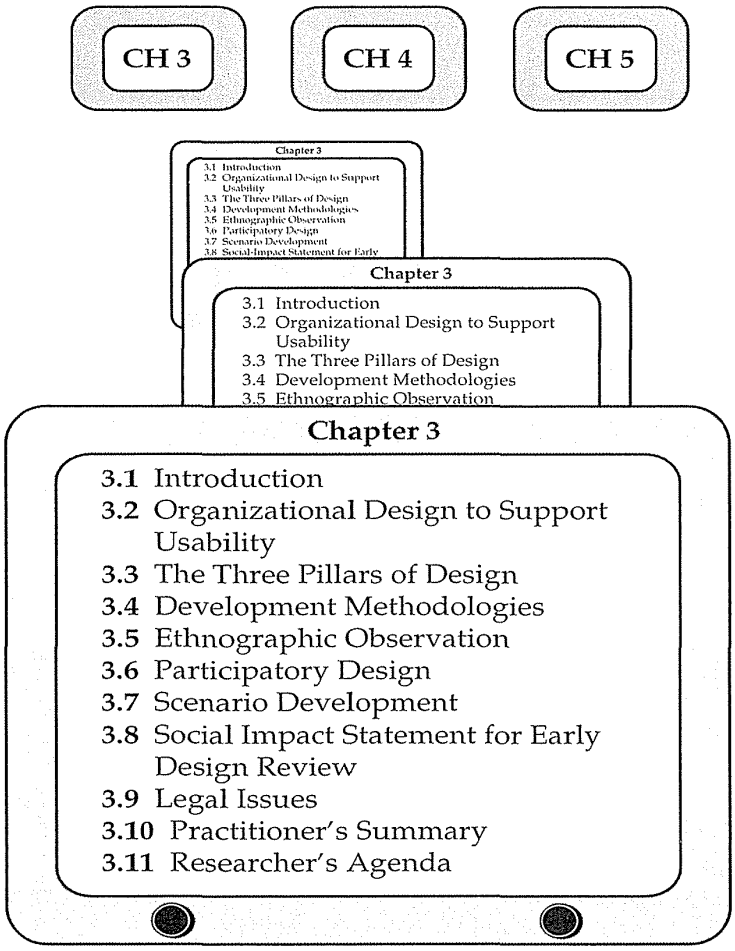
Managing Design Processes

Just as we can assert that no product has ever been created in a single moment of inspiration ... nobody has ever produced a set of requirements for any product in a similarly miraculous manner. These requirements may well begin with an inspirational moment but, almost certainly, the emergent bright idea will be developed by iterative processes of evaluation until it is thought to be worth starting to put pencil to paper. Especially when the product is entirely new, the development of a set of requirements may well depend upon testing initial ideas in some depth.

W. H. Mayall, ***Principles in Design,*** **1979**

The Plan is the generator. Without a plan, you have lack of order and willfulness. The Plan holds in itself the essence of sensation.

Le Corbusier, ***Towards a New Architecture,*** **1931**



3.1 Introduction

In the first decades of computer-software development, technically oriented programmers designed text editors, programming languages, and applications for themselves and their peers. The substantial experience and motivation of these users meant that complex interfaces were accepted and even appreciated. Now, the user population for office automation, home and personal computing, and digital libraries is so vastly different from the original that programmers' intuitions may be inappropriate. Current users are not dedicated to the technology, their background is more tied to workflow, and

their use of computers may be discretionary. Designs should be based on careful observation of current users, refined by thoughtful analysis of task frequencies and sequences, and validated through early, careful, and thorough prototype, usability, and acceptance tests.

In the best designs, the techno-centric style of the past is yielding to a genuine desire to accommodate to the users' skills, goals, and preferences. Designers are seeking direct interaction with users during the design phase, during the development process, and throughout the system lifecycle. Iterative design methods that allow early testing of prototypes, revisions based on feedback from users, and incremental refinements suggested by usability-test administrators are catalysts for high-quality systems. Around the world, *usability engineering* is becoming a recognized discipline with established practices and some standards. The Usability Professionals Association, formed in 1991, has become a respected community with active participation from large corporations and numerous small design, test, and build firms.

The variety of design situations precludes a comprehensive strategy. Managers will have to adapt the strategies offered in this chapter to suit their organization, projects, schedules, and budgets. These strategies begin with the organizational design that gives appropriate emphasis to support usability. Next, are the three pillars of successful user-interface development: guidelines document and process, user-interface software tools, and expert review and usability testing. The Logical User-Centered Interaction Design (LUCID) methodology is a framework for scheduling, on which strategies such as ethnographic observation, participatory design, scenario development, and possibly a Social Impact Statement review can be hung. Finally, legal concerns should be addressed during the design process.

3.2 Organizational Design to Support Usability

Corporate-marketing and customer-assistance departments are becoming more aware of the importance of usability and are a source of constructive encouragement. When competitive products provide similar functionality, usability engineering is vital for product acceptance. Many organizations have created usability laboratories to provide expert reviews and to conduct usability tests of products during development. Outside experts can provide fresh insights, while usability-test subjects perform benchmark tasks in carefully supervised conditions (Whiteside et al., 1988; Klemmer, 1989; Nielsen, 1993; Dumas and Redish, 1993). These and other evaluation strategies are covered in Chapter 4.

Companies may not yet have a chief usability officer (CUO) or a vice president for usability, but they often have user-interface architects and usability engineering managers. High-level commitment helps to promote attention at every level. Organizational awareness can be stimulated by "Usability Day" presentations, internal seminars, newsletters, and awards. Of course, resistance to new techniques and a changing role for software engineers can cause problems in organizations. Organizational change is difficult, but creative leaders blend inspiration and provocation. The high road is to appeal to the desire for quality that most professionals share. When they are shown data on shortened learning times, faster performance, or lower error rates on well-designed interfaces, they are likely to be more sympathetic to applying usability-engineering methods. The low road is to point out the frustration, confusion, and high error rates due to the current complex designs, while citing the successes of competitors who apply usability-engineering methods.

Most large and many small organizations maintain a centralized human-factors group or a usability laboratory as a source of expertise in design and testing techniques (Gould et al., 1991; Nielsen, 1994). However, each project should have its own user-interface architect who develops the necessary skills, manages the work of other people, prepares budgets and schedules, and coordinates with internal and external human-factors professionals when further expertise, references to the literature, or usability tests are required. This dual strategy balances the needs for centralized expertise and decentralized application. It enables professional growth in the user-interface area and in the application domain (for example, in geographic information or imaging systems).

The field has now matured to the point that many projects have grown large in complexity, size, and importance. Role specialization is emerging, as it has in architecture, aerospace, and book design. Eventually, individuals will become highly skilled in specific problems, such as user–interface building tools, graphic-display strategies, voice and audio tone design, and message, or online tutorial writing. Consultation with graphic artists, book designers, advertising copy writers, instructional-textbook authors, or film-animation creators is expected. Perceptive system developers recognize the need to employ psychologists for conducting experimental tests, sociologists for evaluating organizational impact, educational psychologists for refining training procedures, and social workers for guiding user consultants or customer-service personnel.

As design moves to implementation, the choice of user interface building tools is vital to success. These rapidly emerging tools enable designers to build novel systems quickly and support the iterative design–test–refine cycle.

Guidelines documents were originally seen as the answer to usability questions, but they are now appreciated as a broader social process in which the initial compilation is only the first step. The management strategies for

the three Es—enforcement, exemption, enhancement—are only beginning to emerge and to become institutionalized.

The business case for focusing on usability has been made powerfully and repeatedly in the past decade (Mantei and Teorey, 1988; Karat, 1990; Chapanis, 1991). It apparently needs frequent repetition because traditional managers and engineers are often resistant to changes that would bring increased attention to the users' needs. Karat's (1990, 1994) businesslike reports within IBM became influential documents when they were published externally. She reported up to $100 payoffs for each dollar spent on usability, with identifiable benefits in reduced program-development costs, reduced program-maintenance costs, increased revenue due to higher customer satisfaction, and improved user efficiency and productivity. Other economic analyses showed fundamental changes in organizational productivity (as much as 720 percent improvements) when people kept usability in mind from the beginning of development projects (Landauer, 1995). Even minimal application of usability testing followed by correction of 20 of the easiest-to-repair faults improved user efficiency from 19 percent to as much as 80 percent.

Usability engineers and *user-interface architects* are gaining experience in managing organizational change. As attention shifts from software-engineering or management-information systems, battles for control and power manifest themselves in budget and personnel allocations. Well-prepared managers who have a concrete organizational plan, defensible cost–benefit analyses, and practical development methodologies are most likely to be winners.

Design is inherently creative and unpredictable. Interactive system designers must blend a thorough knowledge of technical feasibility with a mystical esthetic sense of what attracts users. Carroll and Rosson (1985) characterize design in this way:

- Design is a *process;* it is not a state and it cannot be adequately represented statically.
- The design process is *nonhierarchical;* it is neither strictly bottom-up nor strictly top-down.
- The process is *radically transformational;* it involves the development of partial and interim solutions that may ultimately play no role in the final design.
- Design intrinsically involves the *discovery of new goals.*

These characterizations of design convey the dynamic nature of the process. But in every creative domain, there can also be discipline, refined techniques, wrong and right methods, and measures of success. Once the

early data collection and preliminary requirements are established, more detailed design and early development can begin. This chapter covers strategies for managing early stages of projects and offers design methodologies. Chapter 4 focuses on evaluation methods.

3.3 The Three Pillars of Design

If standardization can be humanized and made flexible in design and the economics brought to the home owner, the greatest service will be rendered to our modern way of life. It may be really born—this democracy, I mean.

Frank Lloyd Wright, The Natural House, 1954

The three pillars described in this section can help user-interface architects to turn good ideas into a successful system (Fig. 3.1). They are not guaranteed to work, but experience has shown that each pillar can produce an order-of-magnitude speedup in the process and can facilitate the creation of excellent systems.

3.3.1 Guidelines documents and processes

Early in the design process, the user-interface architect should generate, or require other people to generate, a set of working guidelines. Two people might work for one week to produce a 10-page document, or a dozen people might work for two years to produce a 300-page document. One component of Apple's success with the Macintosh was that machine's early and readable guidelines document that provided a clear set of principles for the many applications developers to follow and thus ensured a harmony in design across products. Microsoft's Windows guidelines have also been refined over the years, and they provide a good starting point and an educational experience for many programmers. These and other guidelines documents are referenced and are described briefly in the general reference section at the end of Chapter 1.

Each project has different needs, but guidelines should be considered for

- Words and icons
  - Terminology (objects and actions), abbreviations, and capitalization
  - Character set, fonts, font sizes, and styles (bold, italic, underline)
  - Icons, graphics, and line thickness
  - Use of color, backgrounds, highlighting, and blinking

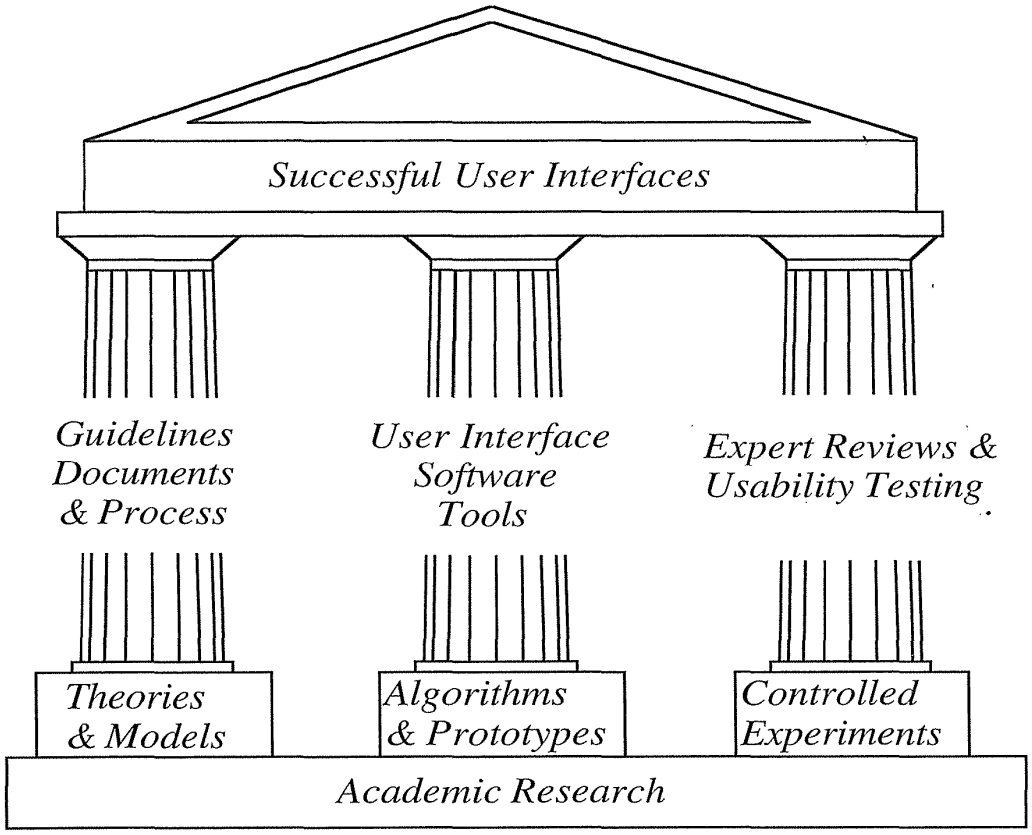


Figure 3.1

The three pillars of successful user-interface development.

- Screen-layout issues
  - Menu selection, form fillin, and dialog-box formats
  - Wording of prompts, feedback, and error messages
  - Justification, whitespace, and margins
  - Data entry and display formats for items and lists
  - Use and contents of headers and footers
- Input and output devices
  - Keyboard, display, cursor control, and pointing devices
  - Audible sounds, voice feedback, touch input, and other special input modes or devices
  - Response times for a variety of tasks
- Action sequences
  - Direct-manipulation clicking, dragging, dropping, and gestures
  - Command syntax, semantics, and sequences

  - Programmed function keys
  - Error handling and recovery procedures
- Training
  - Online help and tutorials
  - Training and reference materials

Guidelines creation should be a social process within an organization to gain visibility and build support. Controversial guidelines (for example, on when to use voice alerts) should be reviewed by colleagues or tested empirically. Procedures should be established to distribute the guidelines, to ensure enforcement, to allow exemptions, and to permit enhancements. Guidelines documents must be a living text that is adapted to changing needs and refined through experience. Acceptance may be increased by a three-level approach of rigid standards, accepted practices, and flexible guidelines. This approach clarifies which items are firmer and which items are susceptible to change.

The creation of a guidelines document (Box 3.1) at the beginning of an implementation project focuses attention on the interface design and provides an opportunity for discussion of controversial issues. When the guideline is adopted by the development team, the implementation proceeds quickly and with few design changes. For large organizations, there may be two or more levels of guidelines to provide organizational identity while allowing projects to have distinctive style and local control of terminology.

3.3.2 User-interface software tools

One difficulty in designing interactive systems is that customers and users may not have a clear idea of what the system will look like when it is done. Since interactive systems are novel in many situations, users may not realize the implications of design decisions. Unfortunately, it is difficult, costly, and time consuming to make major changes to systems once those systems have been implemented.

Even though this problem has no complete solution, some of the more serious difficulties can be avoided if, at an early stage, the customers and users can be given a realistic impression of what the final system will look like (Gould and Lewis, 1985). A printed version of the proposed displays is helpful for pilot tests, but an onscreen display with an active keyboard and mouse is more realistic. The prototype of a menu system may have only one or two paths active, instead of the thousands of paths envisioned for the final system. For a form-fillin system, the prototype may simply show the fields, but may not process them. Prototypes have been developed with simple drawing or word-processing tools, but graphical design environments such as HyperCard and MacroMind Director are widely used. Development envi-

Box 3.1

Recommendations for guidelines documents.

- Provides a social process for developers
- Records decisions for all parties to see
- Promotes consistency and completeness
- Facilitates automation of design
- Allows multiple levels
  - Rigid standards
  - Accepted practices
  - Flexible guidelines
- Announces policies for
  - Enforcement: who reviews?
  - Exemption: who decides?
  - Enhancement: how often?

ronments such as Microsoft's Visual Basic/C++ and Borland's Delphi are easy to learn yet powerful. More sophisticated tools such as Visix's Galaxy and Sun's Java provide cross-platform development and a rich variety of services. These tools are covered in Chapter 5.

3.3.3 Expert reviews and usability testing

Theatrical producers know that previews to critics and extensive rehearsals are necessary to ensure a successful opening night. Early rehearsals may require only one or two performers wearing street clothes; but, as opening night approaches, dress rehearsals with the full cast, props, and lighting are expected. Aircraft designers carry out wind-tunnel tests, build plywood mockups of the cabin layout, construct complete simulations of the cockpit, and thoroughly flight test the first prototype. Similarly, interactive-system designers are now recognizing that they must carry out many small and some large pilot tests of system components before release to customers (Dumas and Redish, 1993). In addition to a variety of expert review methods, tests with the intended users, surveys, and automated analysis tools are proving to be valuable. Procedures vary greatly depending on the goals of the usability study, the number of expected users, the dangers of errors, and the level of investment. Chapter 4 covers expert reviews, usability testing, and other evaluation methods in depth.

3.4 Development Methodologies

Many software development projects fail to achieve their goals. Some estimates suggest that the failure rate is as high as 60 percent, with about 25 percent of projects never being completed and perhaps another 35 percent only achieving partial success. Much of this problem can be traced to lack of attention to design issues during the initial stages of development. Careful attention to user-centered design issues at the early stages of software development has been shown to reduce both development time and cost dramatically. Well-designed systems are less expensive to develop and have lower maintenance costs over their lifetime. They are easier to learn, produce faster performance, reduce user errors substantially, and provide users with a sense of mastery and the confidence to explore features that go beyond the minimum required to get by.

The relationship between software developers and users has not always been a smooth one. Software-engineering development methodologies have helped developers meet budgets and schedules (Boehm, 1988; Sutcliffe and McDermott, 1991; Preece and Rombaugh, 1994; Humphrey, 1995), but have not always provided guidance in developing a usable interface (Chapanis and Budurka, 1990). A number of academics with consulting experience produced a first generation of design methodologies focused on user interface (Hix and Hartson, 1993; Nielsen, 1993). Commercial firms that specialize in user-centered design have built on this foundation and created a second generation of design methodologies.

These business-oriented approaches specify detailed deliverables for the various stages of design and incorporate cost/benefit and return on investment (ROI) analyses to facilitate decision making. In addition to the interface design elements that were basic to the academic systems, the commercial methodologies highlight management strategies used to keep to schedule and budget. Any user-centered design methodology must also mesh with the software-engineering methodology used.

The *Logical User-Centered Interactive Design Methodology* (LUCID, formerly Quality Usability Engineering (QUE)) (Kreitzberg, 1996) identifies six stages (see Table 3.1):

Stage 1: Develop product concept

Stage 2: Perform research and needs analysis

Stage 3: Design concepts and key-screen prototype

Stage 4: Do iterative design and refinement

Stage 5: Implement software

Stage 6: Provide rollout support

In the first stage, a product concept is developed. Surprisingly, many software development efforts are launched without a clear concept of the prod-

Table 3.1

Logical User-Centered Interaction Design Methodology from Cognetics Corporation, Princeton Junction, NJ (Kreitzberg, 1996).

Stage 1: Develop product concept

- Create a high concept.
- Establish business objectives.
- Set up the usability design team.
- Identify the user population.
- Identify technical and environmental issues.
- Produce a staffing plan, schedule, and budget.

Stage 2: Perform research and needs analysis

- Partition the user population into homogeneous segments.
- Break job activities into task units.
- Conduct needs analysis through construction of scenarios and participatory design.
- Sketch the process flow for sequences of tasks.
- Identify major objects and structures which will be used in the software interface.
- Research and resolve technical issues and other constraints.

Stage 3: Design concepts and key-screen prototype

- Create specific usability objectives based on user needs.
- Initiate the guidelines and style guide.
- Select a navigational model and a design metaphor.
- Identify the set of key screens: login, home, major processes.
- Develop a prototype of the key screens using a rapid prototyping tool.
- Conduct initial reviews and usability tests.

Stage 4: Do iterative design and refinement

- Expand key-screen prototype into full system.
- Conduct heuristic and expert reviews.
- Conduct full-scale usability tests.
- Deliver prototype and specification.

Stage 5: Implement software

- Develop standard practices.
- Manage late stage change.
- Develop online help, documentation and tutorials.

Stage 6: Provide rollout support

- Provide training and assistance.
- Perform logging, evaluation, and maintenance.

uct. At the center of the LUCID methodology is creation of a "high concept" for the product—a brief statement that defines the goals, functionality and benefits of the product. For example,

> The new home banking system will provide customers with unified access to their accounts. It will support balance inquiry, management of credit accounts and loans, transfer of funds among accounts, electronic bill payment and investment in the bank's family of mutual funds. The system will provide the customer with year-end accounting for tax purposes.

As part of the product concept stage, project leaders define business objectives, establish the design team, identify environmental, technical or legal constraints, specify the user population, and prepare a project plan and budget. During the first stage, the product concept is illustrated by simple screen sketches (which may be created on paper or on-screen). The goal of these sketches is to convey the system concept to nontechnical users.

With the project plan in place, the design team meets with users to understand their needs and competencies, the business process to be supported and the functional requirements of the system. LUCID uses participatory design sessions to solicit user input, construct workflow scenarios and define the objects that are central to the design.

A distinctive aspect of LUCID is its focus on a key-screen prototype that incorporates the major navigational paths of the system. The key-screen prototype is used to show users the design of the proposed system and allow them to evaluate and refine it. The key-screen prototype is also used for usability testing and heuristic review. Key screens usually evoke strong reactions, generate early participation, and create momentum for the project.

Like most user-centered design methodologies, LUCID employs rapid prototyping and iterative usability testing (Chapter 4). Because rapid prototyping is key to meeting schedule and budget, LUCID relies on user interface building tools (Chapter 5). The prototypes are usually developed by a programmer who is part of the software engineering team. One of this programmer's responsibilities is to identify interface issues that have implications for the technical architecture of the product. When completed and approved by users, the prototype serves as part of the programming specification for the software engineers.

Finally, LUCID describes a phased rollout approach built on theories of organizational change. Project leaders identify barriers to and construct incentives for adoption of the software. The goal is to ensure a positive reception by customers, users, and managers.

As a management strategy, LUCID makes the commitment to user-centered design explicit and highlights the role of usability engineering in software development by focusing on activities, deliverables, and reviews. At each of the LUCID stages, 12 areas of activity are evaluated; each is tied to specified deliverables and timely feedback through reviews:

1. *Product definition*: high concept for managers and marketers
2. *Business case*: pricing, expected revenues, return on investment, competition
3. *Resources*: duration, effort levels, team members, back-up plans
4. *Physical environment*: ergonomic design, physical installation, communication lines
5. *Technical environment*: hardware and software for development and integration
6. *Users*: multiple communities for interviews, user testing, marketing
7. *Functionality*: services provided to users
8. *Prototype*: early paper prototypes, key screens, running prototypes
9. *Usability*: set measurable goals, conduct tests, refine interface and goals
10. *Design guidelines*: modification of existing guidelines, implementation of review process
11. *Content materials*: identification and acquisition of copyrighted text, audio, and video
12. *Documentation, training and help*: specification, development, and testing paper, video, and online versions

The thoroughness of LUCID comes from its validation and refinement in multiple projects. However, each project has special needs, so any design methodology is only a starting point for project management. LUCID is designed to promote an orderly process, with iterations within a stage and predictable progress among stages. The reality is sometimes more complex, especially for novel projects that may require a return to earlier stages for some parts of the design.

3.5 Ethnographic Observation

The early stages of most methodologies include observation of users. Since interface users form a unique culture, ethnographic methods for observing them in the workplace are likely to become increasingly important. An "ethnographer participates, overtly or covertly, in people's daily lives for an extended period of time, watching what happens, listening to what is said, asking questions" (Hammersley and Atkinson, 1983). As ethnographers, user-interface designers gain insight into individual behavior and the organizational context. User-interface designers differ from traditional ethnographers; in addition to understanding their subjects, user-interface designers observe interfaces in use for the purpose of changing and improving those

interfaces. Whereas traditional ethnographers immerse themselves in cultures for weeks or months, user-interface designers need to limit this process to a period of days or even hours, and still to obtain the relevant data needed to influence a redesign (Hughes et al., 1995). Ethnographic methods have been applied to office work (Suchman, 1983), air-traffic control (Bentley et al., 1992), and other domains (Vaske and Grantham, 1989).

The goal of an observation is to obtain the necessary data to influence interface redesign. Unfortunately, it is easy to misinterpret observations, to disrupt normal practice, and to overlook important information. Following a validated ethnographic process reduces the likelihood of these problems. Guidelines for preparing for the evaluation, performing the field study, analyzing the data, and reporting the findings might include the following (Rose et al., 1995):

Preparation

- Understand organization policies and work culture.
- Familiarize yourself with the system and its history.
- Set initial goals and prepare questions.
- Gain access and permission to observe or interview.

Field Study

- Establish rapport with managers and users.
- Observe or interview users in their workplace, and collect subjective and objective quantitative and qualitative data.
- Follow any leads that emerge from the visits.
- Record your visits.

Analysis

- Compile the collected data in numerical, textual, and multimedia databases.
- Quantify data and compile statistics.
- Reduce and interpret the data.
- Refine the goals and the process used.

Reporting

- Consider multiple audiences and goals.
- Prepare a report and present the findings.

These notions seem obvious when stated but they require interpretation and attention in each situation. For example, understanding the differing perceptions that managers and users have about the efficacy of the current interface will alert you to the varying frustrations that each group will

have. For example, managers may complain about the unwillingness of staff to update information promptly, but staff may be resistant to using the interface because the login process takes 6 to 8 minutes. In preparing for one observation, we appreciated that the manager called to warn us that graduate students should not wear jeans because the users were prohibited from doing so. Learning the technical language of the users is also vital for establishing rapport. It is useful to prepare a long list of questions that you can then filter down by focusing on the proposed goals. Awareness of the differences among user communities, such as those mentioned in Section 1.5, will help to make the observation and interview process more effective.

Data collection can include a wide range of subjective impressions that are qualitative or of subjective reactions that are quantitative, such as rating scales or rankings. Objective data can consist of qualitative anecdotes or critical incidents that capture user experiences, or can be quantitative reports about, for example, the number of errors that occur during a one-hour observation of six users. Deciding in advance what to capture is highly beneficial, but remaining alert to unexpected happenings is also valuable. Written report summaries have proved to be valuable, far beyond expectations; in most cases, raw transcripts of every conversation are too voluminous to be useful.

Making the process explicit and planning carefully may seem awkward to many people whose training stems from computing and information technology. However, a thoughtful applied ethnographic process has proved to have many benefits. It can increase trustworthiness and credibility, since designers learn about the complexities of an organization firsthand by visits to the workplace. Personal presence allows designers to develop working relationships with several end users to discuss ideas; most important, the users may consent to be active participants in the design of their new interface.

3.6 Participatory Design

Many authors have urged participatory design strategies (Olson and Ives, 1981; Mumford, 1983; Ives and Olson, 1984; Gould and Lewis, 1985; Gould et al., 1991; Damodaran, 1996), but the concept is controversial. The arguments in favor suggest that more user involvement brings more accurate information about tasks, an opportunity for users to influence design decisions, the sense of participation that builds users' ego investment in successful implementation, and the potential for increased user acceptance of the final system (Baroudi et al., 1986; Greenbaum and Kyng, 1991; Monk et al., 1993).

On the other hand, extensive user involvement may be costly and may lengthen the implementation period, build antagonism with people who are

not involved or whose suggestions are rejected, force designers to compromise their design to satisfy incompetent participants, and simply build opposition to implementation (Ives and Olson, 1984).

Participatory-design experiences are usually positive, and advocates can point to many important contributions that would have been missed without it. People who are resistant might appreciate the somewhat formalized multiple-case-studies *plastic interface for collaborative technology initiatives through video exploration (PICTIVE)* approach (Muller, 1992). Users sketch interfaces, then use slips of paper, pieces of plastic, and tape to create low-fidelity early prototypes. A scenario walkthrough is then recorded on videotape for presentation to managers, users, or other designers. With the right leadership, PICTIVE can effectively elicit new ideas and be fun for all involved (Muller et al., 1993).

Careful selection of users helps to build a successful participatory design experience. A competitive selection increases participants' sense of importance and emphasizes the seriousness of the project. Participants may be asked to commit to repeated meetings and should be told what to expect about their roles and their influence. They may have to learn about the technology and business plans of the organization, and to act as a communication channel to the larger group of users that they represent.

The social and political environment surrounding the implementation of complex interfaces is not amenable to study by rigidly defined methods or controlled experimentation. Social and industrial psychologists are interested in these issues, but dependable research and implementation strategies may never emerge. The sensitive project leader must judge each case on its merits and must decide what is the right level of user involvement. The personalities of the design-team members and of the users are such critical determinants that experts in group dynamics and social psychology may be useful as consultants.

The experienced user-interface architect knows that organizational politics and the preferences of individuals may be more important than the technical issues in governing the success of an interactive system. The warehouse managers who see their positions threatened by an interactive system that provides senior managers with up-to-date information through desktop displays will ensure that the system fails by delaying data entry or by being less than diligent in guaranteeing data accuracy. The interface designer should take into account the effect on users, and should solicit their participation to ensure that all concerns are made explicit early enough to avoid counterproductive efforts and resistance to change. Novelty is threatening to many people, so clear statements about what to expect when can be helpful in reducing anxiety.

3.7 Scenario Development

When a current interface is being redesigned or a well-polished manual system is being automated, there often are available reliable data about the range and distribution of task frequencies and sequences. If current data do not exist, then logging usage can quickly provide insight. When substantial changes are anticipated, such as in business-process re-engineering, or when a novel application is planned, identifying the tasks and estimating their frequencies is more difficult.

A table with user communities listed across the top and tasks listed down the side is helpful. Each box can then be filled in with the relative frequency with which each user performs each task. Another representation tool is a table of task sequences, indicating which tasks follow other tasks. Often, a flowchart or transition diagram helps designers to record and convey the sequences of possible actions. The thickness of the connecting lines indicates the frequency of the transitions.

In less well-defined projects, many designers have found day-in-the-life scenarios helpful to characterize what happens when users perform typical tasks. During the early design stages, data about current performance should be collected to provide a baseline. Information about similar systems can be gathered, and interviews can be conducted with interested parties, such as users and managers (Carroll, 1995).

An early and easy way to describe a novel system is to write scenarios of usage and then, if possible, to act them out as a form of theater. This technique can be especially effective when multiple users must cooperate (for example, in control rooms, cockpits, or financial trading rooms) or multiple physical devices are used (for example, at customer-service desks, medical laboratories, or hotel check-in areas). Scenarios can represent common or emergency situations, with both novice and expert users.

In developing the National Digital Library, the design team began by writing 81 scenarios that portrayed typical needs of potential users. Here is an example:

> *K–16 Users*: A seventh-grade social-studies teacher is teaching a unit on the Industrial Revolution. He wants to make use of primary source material that would illustrate the factors that facilitated industrialization, the manner in which it occurred, and the impact that it had on society and on the built environment. Given his teaching load, he only has about four hours total to locate and package the supplementary material for classroom use.

Other scenarios might describe how users explore a system, such as this optimistic vision, written for the U.S. Holocaust Museum and Education Center:

> A grandmother and her 10- and 12-year old grandsons have visited the museum before. They have returned this time to the Learning Center to explore what life was like in her shtetl in Poland in the 1930s. One grandson eagerly touches the buttons on the welcome screen, and they watch the 45-second video introduction by the museum director. They then select the button on "History before the Holocaust" and choose to view a list of towns. Her small town is not on the list, but she identifies the larger nearby city, and they get a brief textual description, a map of the region, and a photograph of the marketplace. They read about the history of the town and view 15-second videos of the marketplace activity and a Yiddish theater production. They bypass descriptions of key buildings and institutions, choosing instead to read biographies of a famous community leader and a poet. Finally, they select "GuestBook" and add their names to the list of people who have indicated an affiliation with this town. Further up on the list, the grandmother notices the name of a childhood friend from whom she has not heard in 60 years—fortunately, the earlier visitor has left an address.

This scenario was written to give nontechnical museum planners and the Board of Directors an idea of what could be built if funding were provided. Such scenarios are easy for most people to grasp, and they convey design issues such as physical installation (room and seats for three or more patrons with sound isolation) and development requirements (video production for the director's introduction and conversion of archival films to video).

Some scenario writers take a further step and produce a videotape to convey their intentions. There are famous future scenarios, such as Apple's Knowledge Navigator, made in 1988, which produced numerous controversies. It portrayed a professor using voice commands to talk with a bow-tied preppie character on the screen and touch commands to develop ecological simulations. Many viewers enjoyed the tape, but thought that it stepped over the bounds of reality by having the preppie agent recognize the professor's facial expressions, verbal hesitations, and emotional reactions. In 1994, Bruce Tognazzini's Starfire scenario for Sun Microsystems gave his elaborate but realistic impression of a large-screen work environment that supported rich collaborations with remote users. Bill Gates took video scenarios one step further at the November 1994 Comdex show, screening an hour-long police drama set in 2005 to illustrate digital wallets, interactive home TV, educational databases, and medical communications.

3.8 Social Impact Statement for Early Design Review

Interactive systems often have a dramatic impact on large numbers of users. To minimize risks, a thoughtful statement of anticipated impacts circulated among stakeholders can be a useful process for eliciting productive suggestions early in the development, when changes are easiest.

Information systems are increasingly required to provide services by governments, utilities, and publicly regulated industries. However, some critics have strong negative attitudes about modern technologies: "technological evolution is leading to something new: a worldwide interlocked monolithic, technical-political web of unprecedented negative implications. And it is surely creating terrible and possibly catastrophic impacts on the earth" (Mander, 1991).

This negative view does not help us to shape more effective technology or to prevent damage from technology failures. Constructive criticism and guidelines for design could be helpful in reversing the long history of disruptions in telephone, banking, or charge-card systems; dissatisfaction with privacy protection or incorrect credit histories; dislocation through deskilling or layoffs; and deaths from flawed medical instruments. While guarantees of perfection are not possible, policies and processes can be developed that will more often than not lead to satisfying outcomes.

A *social impact statement,* similar to an environmental-impact statement (Battle et al., 1994) might help to promote high-quality systems in government-related applications. Reviews for private-sector corporate projects would be optional and self-administered. Early and widespread discussion can uncover concerns and enable stakeholders to state their positions openly (Ralls, 1994). Of course, there is the danger that these discussions will elevate fears or force designers to make unreasonable compromises, but these risks seem reasonable in a well-managed project. The practicality of writing social impact statements was addressed by Huff (1996), who used them as a teaching tool. An outline for a social impact statement might include these sections (Shneiderman and Rose, 1996):

Describe the new system and its benefits

- Convey the high-level goals of the new system.
- Identify the stakeholders.
- Identify specific benefits.

Address concerns and potential barriers

- Anticipate changes in job functions and potential layoffs.

- Address security and privacy issues.
- Discuss accountability and responsibility for system misuse and failure.
- Avoid potential biases.
- Weigh individual rights versus societal benefits.
- Assess tradeoffs between centralization and decentralization.
- Preserve democratic principles.
- Ensure diverse access.
- Promote simplicity and preserve what works.

Outline the development process

- Present an estimated project schedule.
- Propose process for making decisions.
- Discuss expectations of how stakeholders will be involved.
- Recognize needs for more staff, training, and hardware.
- Propose plan for backups of data and equipment.
- Outline plan for migrating to the new system.
- Describe plan for measuring the success of the new system.

A social impact statement should be produced early enough in the development process to influence the project schedule, system requirements, and budget. It could be developed by the system design team, which might include end users, managers, internal or external software developers, and possibly clients. Even for large systems, the social impact statement should be of a size and complexity that make it accessible to users with relevant background.

After the social impact statement is written, it is evaluated by the appropriate review panel plus managers, other designers, end users, and anyone else who will be affected by the proposed system. Potential review panels include federal government units (for example, General Accounting Organization, Office Personnel Management), state legislatures, regulatory agencies (for example, Securities and Exchange Commission or Federal Aviation Administration), professional societies, and labor unions. The review panel receives the written report, holds public hearings, and requests modifications. Citizen groups also are given the opportunity to present their concerns and to suggest alternatives.

Once the social impact statement is adopted, it must be enforced. A social impact statement documents the intentions for the new system, and the stakeholders need to see that those intentions are backed up by actions. Typically, the review panel is the proper authority for enforcement.

The effort, cost, and time should be appropriate to the project, while facilitating a thoughtful review. The process can offer large improvements by preventing problems that could be expensive to repair, improving privacy protection, minimizing legal challenges, and creating more satisfying work environments. Information-system designers take no Hippocratic Oath, but

pledging themselves to strive for the noble goal of excellence in design can win respect and inspire others.

3.9 Legal Issues

As user interfaces have become prominent, serious legal issues have emerged. Every development process should include a review of legal issues that may affect design, implementation, or marketing.

Privacy is always a concern whenever computers are used to store data or to monitor activity. Medical, legal, financial, military, or certain other data often have to be protected to prevent unapproved access, illegal tampering, inadvertent loss, or malicious mischief. Physical security to prohibit access is fundamental; in addition, privacy protection can involve user-interface mechanisms for controlling password access, file-access control, identity checking, and data verification. Users at a public workstation or kiosks want assurance that their password cannot be seen by other people. Effective protection should provide a high degree of privacy with a minimum of confusion and intrusion into work. Encryption and decryption processes may involve complex dialog boxes to specify keys.

A second concern encompasses safety and reliability. User interfaces for aircraft, automobiles, medical equipment, military systems, or nuclear-reactor control rooms can affect life-or-death decisions. If an air-traffic controller is temporarily confused by the contents of the display, that could lead to disaster. If the user interface for such a system is demonstrated to be difficult to understand, it could leave the designer, developer, and operator open to a law suit alleging improper design. Designers should strive to make high-quality and well-tested interfaces that adhere to state-of-the-art design guidelines. Documentation of testing and usage should be maintained to provide accurate data on actual performance. Unlike architecture or engineering, user-interface design is not yet an established profession with clear standards.

A third issue is copyright protection for software and information (Gilbert, 1990; Computer Science and Telecommunications Board, 1991; Samuelson, 1995; 1996). Software developers who have spent time and money to develop a package are frustrated in their attempts to recover their costs and to make a profit if potential users pirate (i.e., make illegal copies of) the package, rather than buy it. Various technical schemes have been tried to prevent copying, but clever hackers can usually circumvent the barriers. It is unusual for a company to sue an individual for copying a program, but cases have been brought against corporations and universities. Site-license agreements are one solution because they allow copying within a site once the fees have been paid. More complicated situations arise in the context of access to online information. If a customer of an online information service pays for time to access to the data-

base, does the customer have the right to extract and store the retrieved information electronically for later use? Can the customer send an electronic copy to a colleague, or sell a bibliography carefully culled from a large commercial database? Do individuals, their employers, or network operators own the information contained in electronic-mail messages? The emergence of the World Wide Web and efforts to build vast digital libraries have raised the temperature and pace of copyright discussions. Publishers are seeking to protect their intellectual assets, and librarians are torn between their desire to serve patrons and their obligations to publishers. If copyrighted works are disseminated freely, then what incentives will there be for publishers and authors? If it is illegal to transmit any copyrighted work without permission or payment, then science, literature, and other fields will suffer. The fair-use doctrine of limited copying for personal and educational purposes helped cope with the questions raised by photocopying technologies, but the perfect rapid copying and dissemination permitted by the network demands a thoughtful update.

A fourth issue is freedom of speech in electronic environments. Do users have a right to make controversial or potentially offensive statements through electronic mail or newsgroups? Are such statements protected by the First Amendment? Are networks like street corners, where freedom of speech is guaranteed, or are networks like television broadcasting, where community standards must be protected? Should network operators be responsible for or prohibited from eliminating offensive or obscene jokes, stories, or images? Controversy has raged over whether network operators have a right to prohibit electronic-mail messages that are used to organize a rebellion against themselves. Another controversy emerged over whether a network operator has a duty to suppress racist electronic-mail remarks or postings to a bulletin board. If libelous statements are transmitted, can a person sue the network as well as the source?

Other legal concerns include adherence to laws requiring equal access for disabled users and attention to changing laws in countries around the world.

The most controversial issue for user-interface designers is that of copyright and patent protection for user interfaces. When user interfaces comprised coded commands in all-capital letters transmitted via Teletype, there was little that could be protected. But the emergence of artistically designed GUIs with animations and extensive online help has led developers to file for copyright protection. This activity has led to many controversies:

- *What material is eligible for copyright?* Since fonts, lines, boxes, shading, and colors cannot usually be accorded copyrights, some people claim that most interfaces are not protectable. Advocates of strong protection claim that the ensemble of components is a creative work, just like a song that is composed of uncopyrightable notes or a poem of uncopyrightable words. Although standard arrangements, such as the rotated-L format of spreadsheets, are not copyrightable, collections of words, such as the

Lotus 1–2–3 menu tree, have been accepted as copyrightable, but such decisions have later been overturned by higher courts. Apple lost its copyright-infringement suit against Microsoft for the Windows interface, in part because the judge insisted on decomposing the interface into elements rather than looking at the overall look and feel. Maybe the most confusing concept is the separation between ideas (not protectable) and expressions (protectable). Generations of judges and lawyers have wrestled with this issue; they agree only that there is "no bright shining line" between idea and expression, and that the distinction must be decided in each case. Most informed commentators would agree that the idea of working on multiple documents at once by showing multiple windows simultaneously is not protectable, but that specific expressions of windows (border decorations, animations for movement, and so on) is protectable. A key point is that there should be a variety of ways to express a given idea. When there is only one way to express an idea—for example, a circle for the idea of a wedding band—the expression is not protectable.

- *Are copyrights or patents more appropriate for user interfaces?* Traditionally, copyright is used for artistic, literary, and musical expressions, whereas patent is used for functional devices. There are interesting crossovers, such as copyrights for maps, engineering drawings, and decorations on teacups, and patents for software algorithms. In the United States, copyrights are easy to obtain (just put a copyright notice on the user interface and file a copyright application), are rapid, and are not verified. Patents are complex, slow, and costly to obtain, because they must be verified by the U.S. Patent and Trademark Office. Copyrights last 75 years for companies and life plus 50 years for individuals. Patents last for only 17 years but are considered more enforceable. The strength of patent protection has raised concerns over patents that were granted for what appear to be fundamental algorithms for data compression and display management. Copyrights for printed user manuals and online help can also be obtained.
- *What constitutes copyright infringement?* If another developer copies your validly copyrighted user interface exactly, that is clearly a case of infringement. More subtle issues arise when a competitor makes a user interface that has elements strikingly similar, by your judgment, to your own. To win a copyright-infringement case, you must convince a jury of "ordinary observers" that the competitor actually saw your interface and that the other interface is "substantially similar" to yours.
- *Should user interfaces be copyrighted?* There are many respected commentators who believe that user interfaces should not be copyrighted. They contend that user interfaces should be shared and that it would impede progress if developers had to pay for permission for every user-interface feature that they saw and wanted to include in their interface. They claim also that copyrights interfere with beneficial standardiza-

tion and that unnecessary artistic variations would create confusion and inconsistency. Advocates of copyrights for user interfaces wish to recognize creative accomplishments and, by allowing protection, to encourage innovation while ensuring that designers are rewarded for their works. Although ideas are not protectable, specific expressions would have to be licensed from the creator, presumably for a fee, in the same way that each photograph in an art book must be licensed and acknowledged, or each use of a song, play, or quote must be granted permission. Concern over the complexity and cost of this process and the unwillingness of copyright owners to share is legitimate, but the alternative of providing no protection might slow innovation.

In the current legal climate, interface designers must respect existing expressions and would be wise to seek licenses or cooperative agreements to share user interfaces. Placing a copyright notice on the title screen of a system and in user manuals seems appropriate. Of course, proper legal counsel should be obtained.

3.10 Practitioner's Summary

Usability engineering is maturing rapidly, and once-novel ideas have become standard practices. Usability has increasingly taken center stage in organizational and product planning. Development methodologies, such as Cognetics' LUCID, help designers by offering a validated process with predictable schedules and meaningful deliverables. Ethnographic observation can provide information to guide task analysis and to complement carefully supervised participatory design processes. Logs of usage provide valuable data about the task sequences and frequencies. Scenario writing helps to bring common understanding of design goals and is useful for managerial and customer presentations. For interfaces developed by governments, public utilities, and regulated industries, an early social-impact statement can elicit public discussion that is likely to identify problems and produce interfaces that have high overall societal benefits. Designers and managers should obtain legal advice to ensure adherence to laws and protection of intellectual property.

3.11 Researcher's Agenda

Human-interface guidelines are often based on best-guess judgments rather than on experimental data. More experimentation could lead to refined standards that are more complete and dependable, and to more precise knowl-

edge of how much improvement can be expected from a design change. Because of changing technology, we will never have a stable and complete set of guidelines, but the benefits of scientific studies will be enormous in terms of the reliability and quality of decision making about user interfaces. The design processes, ethnographic methods, participatory design activities, scenario writing, and social impact statements are rapidly evolving. Thoughtful case studies of successes and failures would lead to refinement and more widespread application. Creative processes are notoriously difficult to study, but well-documented examples of success stories might inform and inspire.

World Wide Web Resources WWW

Design methods promoted by companies and standards organizations are covered, with information on how to develop style guidelines. References to guidelines documents are included in Chapter 1.

http://www.aw.com/DTUI

References

Baroudi, Jack J., Olson, Margrethe H., and Ives, Blake, An empirical study of the impact of user involvement on system usage and information satisfaction, *Communications of the ACM*, 29, 3 (March 1986), 232–238.

Battle, Jackson, Fischman, Robert, and Squillace, Mark, (1994), *Environmental Law. Volume 1: Environmental Decision making NEPA and the Endangered Species Act*, Anderson Publishing, (1994). World Wide Web version prepared in April 1994 by Robert Fischman, Indiana University, School of Law, Bloomington, IN, http://www.law.indiana.edu/envdec

Bentley, R., Hughes, J., Randall, D., Rodden, T., Sawyer, P., Shapiro, D., and Sommerville, I., Ethnographically-informed systems design for air traffic control, *Proc. CSCW '92—Sharing Perspectives* (1992), 123–129.

Boehm, Barry, A spiral model of software development and enhancement, *IEEE Computer*, 21, 5 (May 1988), 61–72.

Carroll, John, M., *Scenario-Based Design: Envisioning Work and Technology in System Development*, John Wiley and Sons, New York (1995).

Carroll, John M. and Rosson, Mary Beth, Usability specifications as a tool in iterative development. In Hartson, H. Rex (Editor), *Advances in Human–Computer Interaction 1*, Ablex, Norwood, NJ (1985), 1–28.

Chapanis, Alphonse, The business case for human factors in informatics. In Shackel, Brian and Richardson, Simon (Editors), *Human Factors in Informatics Usability*, Cambridge University Press, Cambridge, U.K. (1991), 39–71.

Chapanis, Alphonse and Budurka, William J., Specifying human–computer interface requirements, *Behaviour and Information Technology*, 9, 6 (1990), 479–492.

Computer Science and Telecommunications Board, National Research Council, *Intellectual Property Issues in Software,* National Academy Press, Washington, D.C. (1991).

Damodaran, Leela, User involvement in the systems design process—a practical guide for users, *Behaviour & Information Technology,* 15, 6 (1996), 363–377.

Dumas, Joseph and Redish, Janice, *A Practical Guide to Usability Testing,* Ablex, Norwood, NJ (1993).

Gilbert, Steven W., Information technology, intellectual property, and education, *EDUCOM Review,* 25, (1990), 14–20.

Gould, John, How to design usable systems. In Helander, Martin (Editor), *Handbook of Human–Computer Interaction,* North-Holland, Amsterdam, The Netherlands (1988), 757–789.

Gould, John D., and Lewis, Clayton, Designing for usability: Key principles and what designers think, *Communications of the ACM,* 28, 3 (March 1985), 300–311.

Gould, John D., Boies, Stephen J., and Lewis, Clayton, Making usable, useful productivity-enhancing computer applications, *Communications of the ACM,* 34, 1 (January 1991), 75–85.

Greenbaum, Joan and Kyng, Morten (Editors), *Design at Work: Cooperative Design of Computer Systems,* LEA Publishers, Hillsdale, NJ (1991).

Hammersley, M., and Atkinson, P., *Ethnography Principles and Practice,* Routledge, London (1983).

Hix, Deborah and Hartson, H. Rex, *Developing User Interfaces: Ensuring Usability Through Product and Process,* John Wiley and Sons, New York (1993).

Huff, Chuck, Practical guidance for teaching the Social Impact Statement, *Proc. CQL '96, ACM SIGCAS Symposium on Computers and the Quality of Life* (Feb. 1996), 86–89.

Hughes, J., King, V., Rodden, T., and Anderson, H., The role of ethnography in interactive systems design, *Interactions,* 2, 2 (1995), 56–65.

Humphrey, Watts, *A Discipline for Software Engineering,* Addison-Wesley, Reading, MA (1995).

Ives, Blake, and Olson, Margrethe H., User involvement and MIS success: A review of research, *Management Science,* 30, 5 (May 1984), 586–603.

Karat, Claire-Marie, Cost-benefit analysis of usability engineering techniques, *Proc. Human Factors Society Annual Meeting* (1990), 839–843.

Karat, Claire-Marie, A business case approach to usability. In Bias, Randolph, and Mayhew, Deborah (Editors), *Cost-Justifying Usability,* Academic Press, New York (1994), 45–70.

Klemmer, Edmund T. (Editor), *Ergonomics: Harness the Power of Human Factors in Your Business,* Ablex, Norwood, NJ (1989).

Kreitzberg, Charles, Managing for usability. In Alber, Antone F. (Editor), *Multimedia: A Management Perspective,* Wadsworth, Belmont, CA (1996), 65–88.

Landauer, Thomas K., *The Trouble with Computers: Usefulness, Usability, and Productivity,* MIT Press, Cambridge, MA (1995).

Mander, Jerry, *In the Absence of the Sacred: The Failure of Technology and the Survival of the Indian Nations,* Sierra Club Books, San Francisco, CA (1991).

Mantei, Marilyn and Teorey, Toby, Cost-benefit analysis for incorporating human factors in the software life cycle, *Communications of the ACM*, 31, 4 (1988), 428–439.

Monk, A., Wright, P., Haber, J., and Davenport, L., *Improving Your Human–Computer Interface: A Practical Technique*, Prentice-Hall, Englewood Cliffs, NJ (1993).

Muller, Michael J., Retrospective on a year of participatory design using the PICTIVE technique, *Proc. CHI '92—Human Factors in Computing Systems*, ACM, New York (1992), 455–462.

Muller, M., Wildman, D., and White, E., Taxonomy of PD practices: A brief practitioner's guide, *Communications of the ACM*, 36, 4 (1993), 26–27.

Mumford, Enid, *Designing Participatively*, Manchester Business School, Manchester, U.K. (1983).

Nielsen, Jakob (Editor), Special Issue on Usability Laboratories, *Behaviour & Information Technology*, 13, 1 & 2 (January–April 1994).

Nielsen, Jakob, *Usability Engineering*, Academic Press, New York (1993).

Olson, Margrethe H. and Ives, Blake, User involvement in system design: An empirical test of alternative approaches, *Information and Management*, 4, (1981), 183–195.

Preece, Jenny and Rombach, Dieter, A taxonomy for combining Software Engineering (SE) and Human–Computer Interaction (HCI) measurement approaches: Towards a common framework, *International Journal of Human–Computer Studies*, 41, 4 (1994), 553–583.

Ralls, Scott, *Integrating Technology with Workers in the New American Workplace*, U.S. Department of Labor, Office of the American Workplace, Washington, D.C. (1994).

Rose, Anne, Plaisant, Catherine, and Shneiderman, Ben, Using ethnographic methods in user interface re-engineering, *Proc. DIS '95: Symposium on Designing Interactive Systems*, ACM Press, New York (August 1995), 115–122.

Samuelson, Pamela, Copyright and digital libraries, *Communications of the ACM*, 38, 3 (1995), 15–21, 110.

Samuelson, Pamela, Legal protection for database contents, *Communications of the ACM*, 39, 12 (1996), 17–23.

Shneiderman, Ben and Rose, Anne, Social impact statements: Engaging public participation in information technology design, *Proc. CQL '96, ACM SIGCAS Symposium on Computers and the Quality of Life* (Feb. 1996), 90–96.

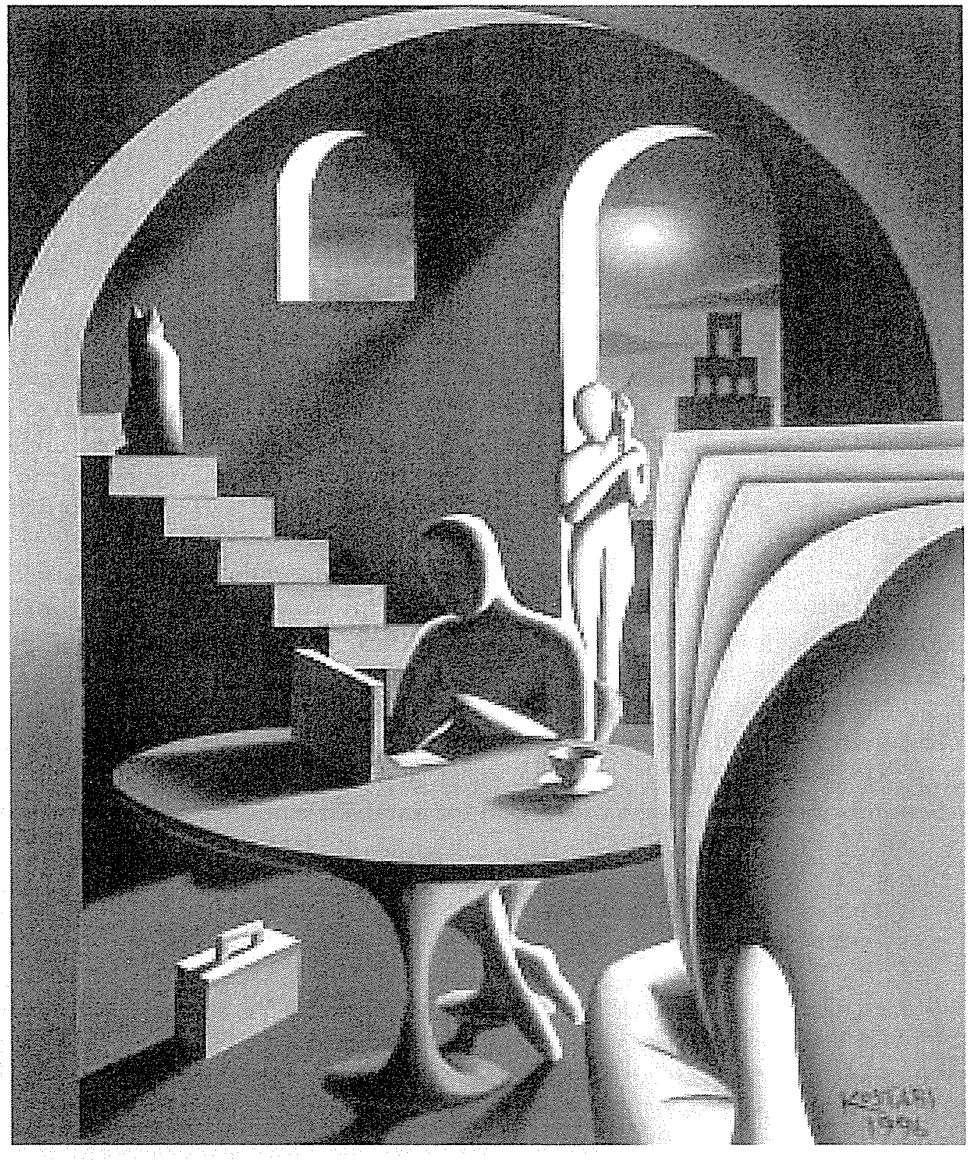
Sutcliffe, A. G. and McDermott, M., Integrating methods of human–computer interface design with structured systems development, *International Journal of Man–Machine Studies*, 34, 5 (1991), 631–656.

Thomas, John C., Organizing for human factors. In Vassiliou, Y. (Editor), *Human Factors in Interactive Computer Systems*, Ablex, Norwood, NJ (1984), 29–46.

Suchman, L., Office procedure as practical action: Models of work and system design, *ACM Transactions on Office Information Systems*, 1, 4 (1983), 320–328.

Vaske, Jerry and Grantham, Charles, *Socializing the Human–Computer Environment*, Ablex, Norwood, NJ (1989).

Whiteside, John, Bennett, John, and Holtzblatt, Karen, Usability engineering: Our experience and evolution. In Helander, Martin (Editor), *Handbook of Human–Computer Interaction*, North-Holland, Amsterdam, The Netherlands (1988), 791–817.



Mark Kostabi, *Gumball Market*, 1996

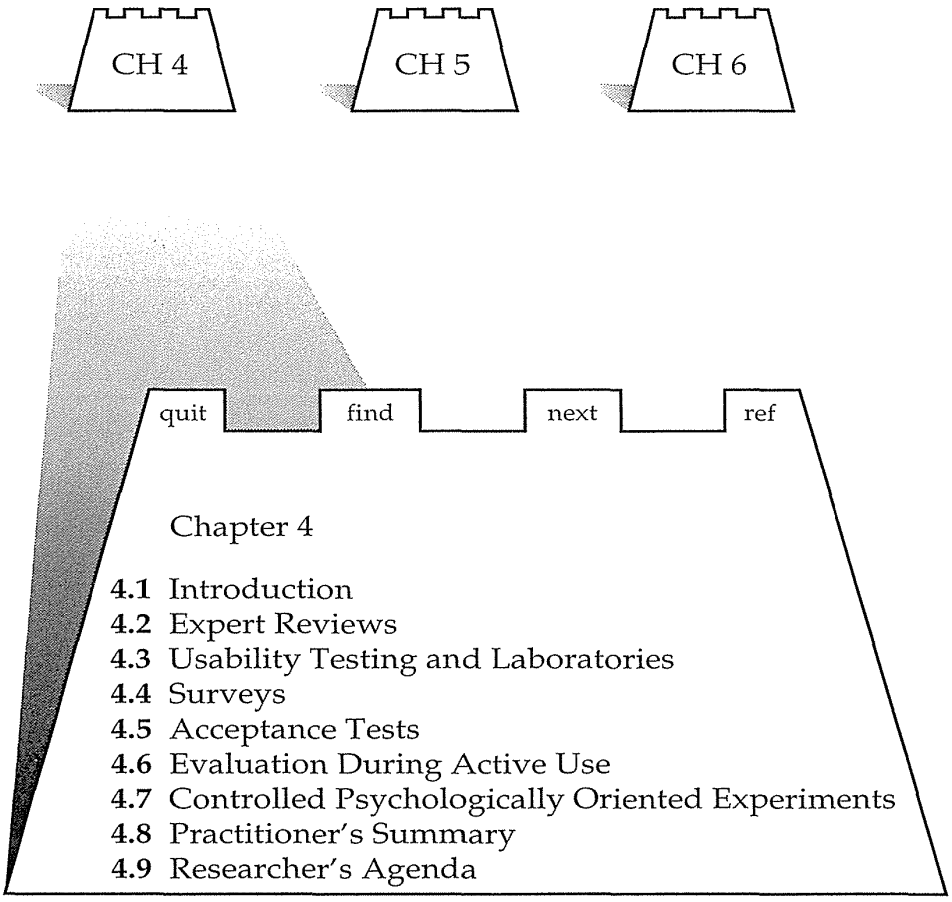
CHAPTER

4

Expert Reviews, Usability Testing, Surveys, and Continuing Assessments

The test of what is real is that it is hard and rough.
. . . What is pleasant belongs in dreams.

Simone Weil, ***Gravity and Grace,*** **1947**



4.1 Introduction

Designers can become so entranced with their creations that they may fail to evaluate those objects adequately. Experienced designers have attained the wisdom and humility to know that extensive testing is a necessity. If feedback is the "breakfast of champions," then testing is the "dinner of the gods." However, careful choices must be made from the large menu of evaluation possibilities to create a balanced meal.

The determinants of the evaluation plan include (Nielsen, 1993; Hix and Hartson, 1993; Preece et al., 1994; Newman and Lamming, 1995)

- Stage of design (early, middle, late)
- Novelty of project (well defined versus exploratory)
- Number of expected users

- Criticality of the interface (for example, life-critical medical system versus museum-exhibit support)
- Costs of product and finances allocated for testing
- Time available
- Experience of the design and evaluation team

The range of evaluation plans might be from an ambitious two-year test with multiple phases for a new national air-traffic–control system to a three-day test with six users for a small internal accounting system. The range of costs might be from 10 percent of a project down to 1 percent.

A few years ago, it was just a good idea to get ahead of the competition by focusing on usability and doing testing, but now the rapid growth of interest in usability means that failure to test is risky indeed. The dangers are not only that the competition has strengthened, but also that customary engineering practice now requires adequate testing. Failure to perform and document testing could lead to failed contract proposals or malpractice lawsuits from users when errors arise. At this point, it is irresponsible to bypass some form of usability testing.

One troubling aspect of testing is the uncertainty that remains even after exhaustive testing by multiple methods. Perfection is not possible in complex human endeavors, so planning must include continuing methods to assess and repair problems during the lifecycle of an interface. Second, even though problems may continue to be found, at some point a decision has to be made about completing prototype testing and delivering the product. Third, most testing methods will account appropriately for normal usage, but performance with high levels of input such as in nuclear-reactor–control or air-traffic–control emergencies is extremely difficult to test. Development of testing methods to deal with stressful situations and even with partial equipment failures will have to be undertaken as user interfaces are developed for an increasing number of life-critical applications.

The Usability Professionals Association was founded in 1991 to exchange information among workers in this arena. The annual conference focuses attention on forms of usability evaluations and provides a forum for exchanges of ideas among the more than 4000 members.

4.2 Expert Reviews

While informal demos to colleagues or customers can provide some useful feedback, more formal expert reviews have proved to be effective (Nielsen and Mack, 1994). These methods depend on having experts available on staff or as consultants, whose expertise may be in the application or user-interface domains. Expert reviews can be conducted on short notice and rapidly.

Expert reviews can occur early or late in the design phase, and the outcomes can be a formal report with problems identified or recommendations for changes. Alternatively, the expert review could result in a discussion with or presentation to designers or managers. Expert reviewers should be sensitive to the design team's ego involvement and professional skill, so suggestions should be made cautiously: It is difficult for someone just freshly inspecting a system to understand all the design rationale and development history. The reviewer notes possible problems for discussion with the designers, but solutions generally should be left for the designers to produce. Expert reviews usually entail half day to one week, although a lengthy training period may be required to explain the task domain or operational procedures. It may be useful to have the same as well as fresh expert reviewers as the project progresses. There are a variety of expert-review methods from which to choose:

- *Heuristic evaluation* The expert reviewers critique an interface to determine conformance with a short list of design heuristics such as the eight golden rules (Chapter 2). It makes an enormous difference if the experts are familiar with the rules and are able to interpret and apply them.
- *Guidelines review* The interface is checked for conformance with the organizational or other guidelines document. Because guidelines documents may contain a thousand items, it may take the expert reviewers some time to master the guidelines, and days or weeks to review a large system.
- *Consistency inspection* The experts verify consistency across a family of interfaces, checking for consistency of terminology, color, layout, input and output formats, and so on within the interface as well as in the training materials and online help.
- *Cognitive walkthrough* The experts simulate users walking through the interface to carry out typical tasks. High-frequency tasks are a starting point, but rare critical tasks, such as error recovery, also should be walked through. Some form of simulating the day in the life of the user should be part of expert-review process. Cognitive walkthroughs were developed for interfaces that can be learned by exploratory browsing (Wharton et al., 1994), but they are useful even for interfaces that require substantial training. An expert might try the walkthrough privately and explore, but then there also would be a group meeting with designers, users, or managers to conduct the walkthrough and to provoke a discussion. This public walkthrough is based on the successful code walkthroughs promoted in software engineering (Yourdon, 1989).
- *Formal usability inspection* The experts hold courtroom-style meeting, with a moderator or judge, to present the interface and to discuss its merits and weaknesses. Design-team members may rebut the evidence about problems in an adversarial format. Formal usability inspections

can be educational experiences for novice designers and managers, but they may take longer to prepare and more personnel to carry out than do other types of review.

Expert reviews can be scheduled at several points in the development process when experts are available and when the design team is ready for feedback. The number of expert reviews will depend on the magnitude of the project and on the amount of resources allocated.

Comparative evaluation of expert-review methods and usability-testing methods is difficult because of the many uncontrollable variables; however, the studies that have been conducted provide evidence for the benefits of expert reviews (Jeffries et al., 1991; Karat et al. 1992). Different experts tend to find different problems in an interface, so three to five expert reviewers can be highly productive, as can complementary usability testing.

Expert reviewers should be placed in the situation most similar to the one that intended users will experience. The expert reviewers should take training courses, read manuals, take tutorials, and try the system in as close as possible to a realistic work environment, complete with noise and distractions. In addition, expert reviewers may also retreat to a quieter environment for detailed review of each screen.

Getting a *bird's-eye view* of an interface by studying a full set of printed screens laid out on the floor or pinned to walls has proved to be enormously fruitful in detecting inconsistencies and spotting unusual patterns.

The dangers with expert reviews are that the experts may not have an adequate understanding of the task domain or user communities. Experts come in many flavors, and conflicting advice can further confuse the situation (cynics say, "For every PhD, there is an equal and opposite PhD"). To strengthen the possibility of successful expert review, it helps to chose knowledgeable experts who are familiar with the project situation and who have a long-term relationship with the organization. These people can be called back to see the results of their intervention, and they can be held accountable. Moreover, even experienced expert reviewers have great difficulty knowing how typical users, especially first-time users, will behave.

4.3 Usability Testing and Laboratories

The emergence of usability testing and laboratories since the early 1980s is an indicator of the profound shift in attention to user needs. Traditional managers and developers resisted at first, saying that usability testing seemed like a nice idea, but that time pressures or limited resources prevented them from trying it. As experience grew and successful projects gave credit to the testing process, demand swelled and design teams began to compete for the scarce

resource of the usability-laboratory staff. Managers came to realize that having a usability test on the schedule was a powerful incentive to complete a design phase. The usability-test report provided supportive confirmation of progress and specific recommendations for changes. Designers sought the bright light of evaluative feedback to guide their work, and managers saw fewer disasters as projects approached delivery dates. The remarkable surprise was that usability testing not only sped up many projects, but also produced dramatic cost savings (Gould, 1988; Gould et al., 1991; Karat, 1994).

Usability-laboratory advocates split from their academic roots as these practitioners developed innovative approaches that were influenced by advertising and market research. While academics were developing controlled experiments to test hypotheses and support theories, practitioners developed usability-testing methods to refine user interfaces rapidly. Controlled experiments have at least two treatments and seek to show statistically significant differences; usability tests are designed to find flaws in user interfaces. Both strategies use a carefully prepared set of tasks, but usability tests have fewer subjects (maybe as few as three), and the outcome is a report with recommended changes, as opposed to validation or rejection of hypotheses. Of course, there is a useful spectrum of possibilities between rigid controls and informal testing, and sometimes a combination of approaches is appropriate.

The movement toward usability testing stimulated the construction of usability laboratories (Dumas and Redish, 1993; Nielsen, 1993). Many organizations spent modest sums to build a single usability laboratory, while IBM built an elaborate facility in Boca Raton, Florida, with 16 laboratories in a circular arrangement with a centralized database for logging usage and recording performance. Having a physical laboratory makes an organization's commitment to usability clear to employees, customers, and users (Nielsen, 1994) (Fig. 4.1). A typical modest usability laboratory would have two 10- by 10-foot areas, one for the participants to do their work and another, divided by a half-silvered mirror, for the testers and observers (designers, managers, and customers) (Fig. 4.2). IBM was an early leader in developing usability laboratories, Microsoft started later, but embraced the idea forcefully, and hundreds of software-development companies have followed suit. A consulting community that will do usability testing for hire also has emerged.

The usability laboratory is typically staffed by one or more people with expertise in testing and user-interface design, who may serve 10 to 15 projects per year throughout the organization. The laboratory staff meet with the user-interface architect or manager at the start of the project to make a test plan with scheduled dates and budget allocations. Usability-laboratory staff participate in early task analysis or design reviews, provide information on software tools or literature references, and help to develop the set of tasks for the usability test. Two to six weeks before the usability test, the detailed test plan is developed, comprising the list of tasks, plus subjective satisfac-

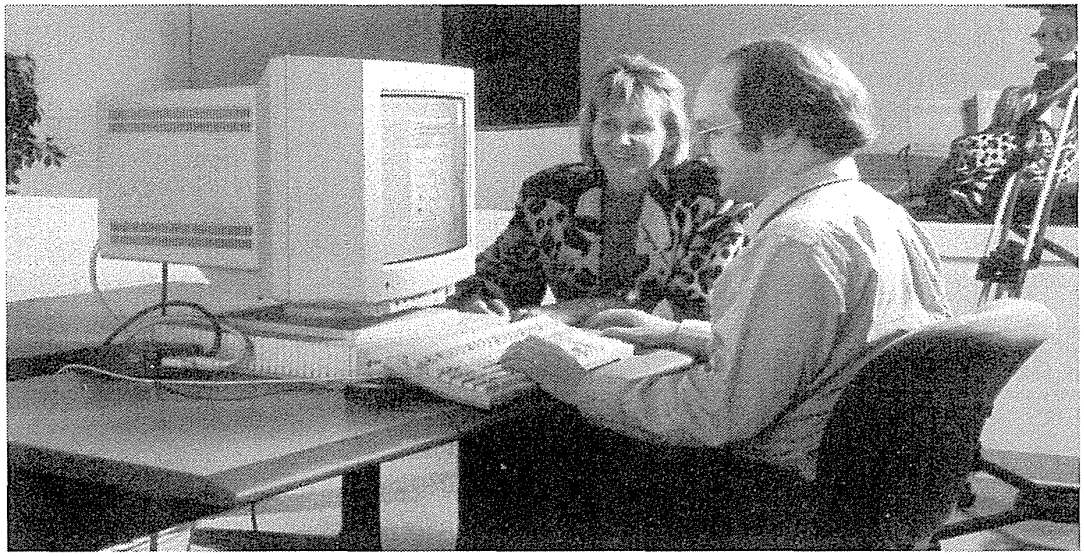


Figure 4.1

Usability lab test, with subject and observer seated at a workstation. Video recorders capture the user's actions and the contents of the screens, while microphones capture thinking-aloud comments. (Used with permission of Sun Microsystems, Mountain View, CA.)

tion and debriefing questions. The number, types, and source of participants are identified—sources, for example, might be customer sites, temporary personnel agencies, or advertisements placed in newspapers. A pilot test of the procedures, tasks, and questionnaires, with one to three subjects is conducted one week ahead of time, while there is still time for changes. This stereotypic preparation process can be modified in many ways to suit each project's unique needs.

After changes are approved, participants are chosen to represent the intended user communities, with attention to background in computing, experience with the task, motivation, education, and ability with the natural language used in the interface. Usability-laboratory staff also must control for physical concerns (such as eyesight, left- versus right-handedness, age, and gender), and for other experimental conditions (such as time of day, day of week, physical surroundings, noise, room temperature, and level of distractions).

Participants should always be treated with respect and should be informed that it is not *they* who are being tested; rather, it is the software and user interface that are under study. They should be told about what they will be doing (for example, typing text into a computer, creating a drawing using a mouse, or getting information from a touchscreen kiosk) and how long they will be expected to stay. Participation should always be voluntary, and

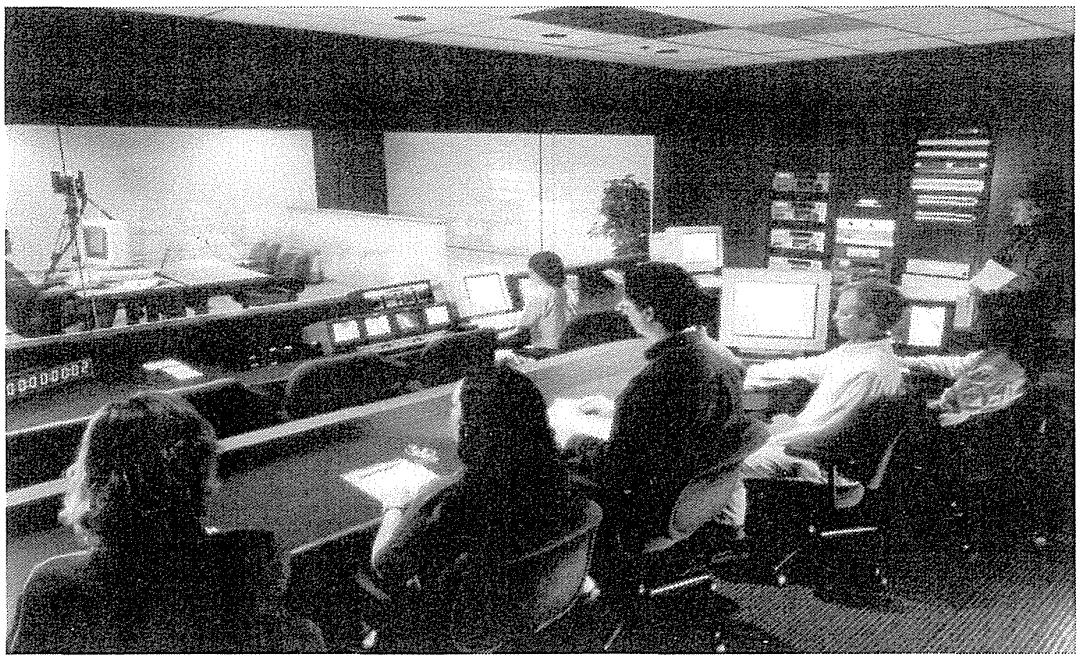


Figure 4.2

Usability lab control room, with test controllers and observers watching the subject through a half-silvered window. Video controls allow zooming and panning to focus on user actions. (Used with permission of Sun Microsystems, Mountain View, CA.)

informed consent should be obtained. Professional practice is to ask all subjects to read and sign a statement like this one:

- I have freely volunteered to participate in this experiment.
- I have been informed in advance what my task(s) will be and what procedures will be followed.
- I have been given the opportunity to ask questions and have had my questions answered to my satisfaction.
- I am aware that I have the right to withdraw consent and to discontinue participation at any time, without prejudice to my future treatment.
- My signature below may be taken as affirmation of all the above statements; it was given prior to my participation in this study.

An effective technique during usability testing is to invite users to think *aloud* about what they are doing. The designer or tester should be supportive of the participants, not taking over or giving instructions, but prompting and listening for clues about how they are dealing with the interface. After a suitable

time period for accomplishing the task list—usually one to three hours—the participants can be invited to make general comments or suggestions, or to respond to specific questions. The informal atmosphere of a thinking-aloud session is pleasant, and often leads to many spontaneous suggestions for improvements. In their efforts to encourage thinking aloud, some usability laboratories found that having two participants working together produces more talking, as one participant explains procedures and decisions to the other.

Videotaping participants performing tasks is often valuable for later review and for showing designers or managers the problems that users encounter (Lund, 1985). Reviewing videotapes is a tedious job, so careful logging and annotation during the test is vital to reduce the time spent finding critical incidents (Harrison, 1991). Participants may be anxious about the video cameras at the start of the test, but within minutes they usually focus on the tasks and ignore the videotaping. The reactions of designers to seeing videotapes of users failing with their system is sometimes powerful and may be highly motivating. When designers see subjects repeatedly picking the wrong menu item, they realize that the label or placement needs to be changed. Most usability laboratories have acquired or developed software to facilitate logging of user activities (typing, mousing, reading screens, reading manuals, and so on) by observers with automatic time stamping.

At each design stage, the interface can be refined iteratively, and the improved version can be tested. It is important to fix quickly even small flaws, such as of spelling errors or inconsistent layout, since they influence user expectations.

Many variant forms of usability testing have been tried. Nielsen's (1993) *discount usability engineering*, which advocates quick and dirty approaches to task analysis, prototype development, and testing, has been widely influential because it lowered the barriers to newcomers.

Field tests attempt to put new interfaces to work in realistic environments for a fixed trial period. Field tests can be made more fruitful if logging software is used to capture error, command, and help frequencies, plus productivity measures. Portable usability laboratories with videotaping and logging facilities have been developed to support more thorough field testing. A different kind of field testing supplies users with test versions of new software. The largest field test of all time was probably the beta-testing of Microsoft's Windows 95, in which reportedly 400,000 users internationally received early versions and were asked to comment.

Early usability studies can be conducted using paper mockups of screen displays to assess user reactions to wording, layout, and sequencing. A test administrator plays the role of the computer by flipping the pages while asking a participant user to carry out typical tasks. This informal testing is inexpensive and rapid, and usually is productive.

Game designers pioneered the *can-you-break-this* approach to usability testing by providing energetic teenagers with the challenge of trying to beat

new games. This destructive testing approach, in which the users try to find fatal flaws in the system or otherwise to destroy it, has been used in other projects and should be considered seriously. Software purchasers have little patience with flawed products and the cost of sending out tens of thousands of replacement disks is one that few companies can bear.

Competitive usability testing can be used to compare a new interface to previous versions or to similar products from competitors. This approach is close to a controlled experimental study, and staff must be careful to construct parallel sets of tasks and to counterbalance the order of presentation of the interfaces. Within subjects designs seem more powerful because participants can make comparisons between the competing interfaces, so fewer participants are needed, although they will each be needed for a longer time period.

For all its success, usability testing does have at least two serious limitations: It emphasizes first-time usage and has limited coverage of the interface features. Since usability tests are usually two to four hours, it is difficult to ascertain how performance will be after a week or a month of regular usage. Within the typical two to four hours of a usability test, the participants may get to use only a small fraction of the features, menus, dialog boxes, or help screens. These and other concerns have led design teams to supplement usability testing with the varied forms of expert reviews.

4.4 Surveys

Written user surveys are a familiar, inexpensive, and generally acceptable companion for usability tests and expert reviews. Managers and users grasp the notion of surveys, and the typically large numbers of respondents (hundreds to thousands of users) offer a sense of authority compared to the potentially biased and highly variable results from small numbers of usability-test participants or expert reviewers. The keys to successful surveys are clear goals in advance and then development of focused items that help to attain those goals. Experienced surveyors know that care is also needed during administration and data analysis (Oppenheim, 1992).

A survey form should be prepared, reviewed among colleagues, and tested with a small sample of users before a large-scale survey is conducted. Similarly, statistical analyses (beyond means and standard deviations) and presentations (histograms, scatterplots, and so on) should also be developed before the final survey is distributed. In short, directed activities are more successful than unplanned statistical-gathering expeditions (no wild goose chases, please). My experience is that directed activities also seem to provide the most fertile frameworks for unanticipated discoveries.

Survey goals can be tied to the components of the OAI model of interface design (see Section 2.3). Users could be asked for their subjective impressions about specific aspects of the interface, such as the representation of

- Task domain objects and actions
- Interface domain metaphors and action handles
- Syntax of inputs and design of displays

Other goals would be to ascertain the user's

- Background (age, gender, origins, education, income)
- Experience with computers (specific applications or software packages, length of time, depth of knowledge)
- Job responsibilities (decision-making influence, managerial roles, motivation)
- Personality style (introvert versus extravert, risk taking versus risk averse, early versus late adopter, systematic versus opportunistic)
- Reasons for not using an interface (inadequate services, too complex, too slow)
- Familiarity with features (printing, macros, shortcuts, tutorials)
- Feelings after using an interface (confused versus clear, frustrated versus in control, bored versus excited)

Online surveys avoid the cost and effort of printing, distributing, and collecting paper forms. Many people prefer to answer a brief survey displayed on a screen, instead of filling in and returning a printed form, although there is a potential bias in the self-selected sample. One survey of World Wide Web utilization generated more than 13,000 respondents. So that costs are kept low, surveys might be administered to only a fraction of the user community.

In one survey, users were asked to respond to eight statements according to the following commonly used scale:

1. Strongly agree
2. Agree
3. Neutral
4. Disagree
5. Strongly disagree

The items in the survey were these:

1. I find the system commands easy to use.
2. I feel competent with and knowledgeable about the system commands.
3. When writing a set of system commands for a new application, I am confident that they will be correct on the first run.

4. When I get an error message, I find that it is helpful in identifying the problem.
5. I think that there are too many options and special cases.
6. I believe that the commands could be substantially simplified.
7. I have trouble remembering the commands and options, and must consult the manual frequently.
8. When a problem arises, I ask for assistance from someone who really knows the system.

This list of questions can help designers to identify problems users are having, and to demonstrate improvement to the interface as changes are made in training, online assistance, command structures, and so on; progress is demonstrated by improved scores on subsequent surveys.

In a study of error messages in text-editor usage, users had to rate the messages on 1-to-7 scales:

Hostile	1 2 3 4 5 6 7	Friendly
Vague	1 2 3 4 5 6 7	Specific
Misleading	1 2 3 4 5 6 7	Beneficial
Discouraging	1 2 3 4 5 6 7	Encouraging

If precise—as opposed to general—questions are used in surveys, then there is a greater chance that the results will provide useful guidance for taking action.

Coleman and Williges (1985) developed a set of bipolar semantically anchored items (pleasing versus irritating, simple versus complicated, concise versus redundant) that asked users to describe their reactions to using a word processor. Another approach is to ask users to evaluate aspects of the interface design, such as the readability of characters, the meaningfulness of command names, or the helpfulness of error messages. If users rate as poor one aspect of the interactive system, the designers have a clear indication of what needs to be redone.

The *Questionnaire for User Interaction Satisfaction* (QUIS) was developed by Shneiderman and was refined by Norman and Chin (Chin et al., 1988) (http://www.lap.umd.edu/QUISFolder/quisHome.html). It was based on the early versions of the OAI model and therefore covered interface details, such as readability of characters and layout of displays; interface objects, such as meaningfulness of icons; interface actions, such as shortcuts for frequent users; and task issues, such as appropriate terminology or screen sequencing. It has proved useful in demonstrating the benefits of improvements to a videodisc-retrieval program, in comparing two Pascal programming environments, in assessing word processors, and in setting requirements for redesign of an online public-access library catalog. We have

since applied QUIS in many projects with thousands of users and have created new versions that include items relating to website design and videoconferencing. The University of Maryland Office of Technology Liaison (College Park, Maryland 20742; (301) 405-4209) licenses QUIS in electronic and paper forms to over a hundred organizations internationally, in addition to granting free licenses to student researchers. The licensees have applied QUIS in varied ways, sometimes using only parts of QUIS or adding domain-specific items.

Table 4.1 contains the long form that was designed to have two levels of questions: general and detailed. If participants are willing to respond to every item, then the long-form questionnaire can be used. If participants are not likely to be patient, then only the general questions in the short form need to be asked.

Other scales include the Post-Study System Usability Questionnaire, developed by IBM, which has 48 items that focus on system usefulness, information quality, and interface quality (Lewis, 1995). The Software Usability Measurement Inventory contains 50 items designed to measure users' perceptions of their effect, efficiency, and control (Kirakowski and Corbett, 1993).

4.5 Acceptance Tests

For large implementation projects, the customer or manager usually sets objective and measurable goals for hardware and software performance. Many authors of requirements documents are even so bold as to specify mean time between failures, as well as the mean time to repair for hardware and, in some cases, for software. More typically, a set of test cases is specified for the software, with possible response-time requirements for the hardware-software combination. If the completed product fails to meet these acceptance criteria, the system must be reworked until success is demonstrated.

These notions can be neatly extended to the human interface. Explicit acceptance criteria should be established when the requirements document is written or when a contract is offered.

Rather than the vague and misleading criterion of "user friendly," measurable criteria for the user interface can be established for the following:

- Time for users to learn specific functions
- Speed of task performance
- Rate of errors by users
- User retention of commands over time
- Subjective user satisfaction

Table 4.1

Questionaire for User Interaction Satisfaction (© University of Maryland, 1997)

Identification number: _____ System: _____ Age: _____ Gender: __ male __ female

PART 1: System Experience

1.1 How long have you worked on this system?

___ less than 1 hour
___ 1 hour to less than 1 day
___ 1 day to less than 1 week
___ 1 week to less than 1 month
___ 1 month to less than 6 months
___ 6 months to less than 1 year
___ 1 year to less than 2 years
___ 2 years to less than 3 years
___ 3 years or more

1.2 On the average, how much time do you spend per week on this system?

___ less than one hour
___ one to less than 4 hours
___ 4 to less than 10 hours
___ over 10 hours

PART 2: Past Experience

2.1 How many operating systems have you worked with?

___ none
___ 1
___ 2
___ 3–4
___ 5–6
___ more than 6

2.2 Of the following devices, software, and systems, check those that you have personally used and are familiar with:

___ computer terminal
___ color monitor
___ CD-ROM drive
___ track ball
___ graphics tablet
___ scanners
___ spreadsheet software
___ voice recognition
___ CAD computer aided design
___ personal computer
___ touch screen
___ keyboard
___ joy stick
___ head mounted display
___ word processor
___ database software
___ video editing systems
___ rapid prototyping systems
___ lap top computer
___ floppy drive
___ mouse
___ pen based computing
___ modems
___ graphics software
___ computer games
___ internet
___ e-mail

PART 3: Overall User Reactions

Please circle the numbers which most appropriately reflect your impressions about using this computer system. Not Applicable = NA.

3.1 Overall reactions to the system: terrible ... wonderful
1 2 3 4 5 6 7 8 9 NA

3.2 frustrating ... satisfying
1 2 3 4 5 6 7 8 9 NA

3.3 dull ... stimulating
1 2 3 4 5 6 7 8 9 NA

3.4 difficult ... easy
1 2 3 4 5 6 7 8 9 NA

3.5 inadequate power ... adequate power
1 2 3 4 5 6 7 8 9 NA

3.6 rigid ... flexible
1 2 3 4 5 6 7 8 9 NA

Table 4.1 (continued)

PART 4: Screen

4.1 Characters on the computer screen	hard to read									easy to read	
	1	2	3	4	5	6	7	8	9		NA
4.1.1 Image of characters	fuzzy									sharp	
	1	2	3	4	5	6	7	8	9		NA
4.1.2 Character shapes (fonts)	barely legible									very legible	
	1	2	3	4	5	6	7	8	9		NA
4.2 Highlighting on the screen	unhelpful									helpful	
	1	2	3	4	5	6	7	8	9		NA
4.2.1 Use of reverse video	unhelpful									helpful	
	1	2	3	4	5	6	7	8	9		NA
4.2.2 Use of blinking	unhelpful									helpful	
	1	2	3	4	5	6	7	8	9		NA
4.2.3 Use of bolding	unhelpful									helpful	
	1	2	3	4	5	6	7	8	9		NA
4.3 Screen layouts were helpful	never									always	
	1	2	3	4	5	6	7	8	9		NA
4.3.1 Amount of information that can be displayed on screen	inadequate									adequate	
	1	2	3	4	5	6	7	8	9		NA
4.3.2 Arrangement of information can be displayed on screen	illogical									logical	
	1	2	3	4	5	6	7	8	9		NA
4.4 Sequence of screens	confusing									clear	
	1	2	3	4	5	6	7	8	9		NA
4.4.1 Next screen in a sequence	unpredictable									predictable	
	1	2	3	4	5	6	7	8	9		NA
4.4.2 Going back to the previous screen	impossible									easy	
	1	2	3	4	5	6	7	8	9		NA
4.4.3 Progression of work related tasks	confusing									clearly marked	
	1	2	3	4	5	6	7	8	9		NA

Please write your comments about the screens here:

PART 5: Terminology and System Information

5.1 Use of terminology throughout system	inconsistent									consistent	
	1	2	3	4	5	6	7	8	9		NA
5.1.2 Work related terminology	inconsistent									consistent	
	1	2	3	4	5	6	7	8	9		NA
5.2.3 Computer terminology	inconsistent									consistent	
	1	2	3	4	5	6	7	8	9		NA

Table 4.1 (continued)

5.2 Terminology relates well to the work you are doing?	never 1 2 3 4 5 6 7 8 9 always	NA
5.2.1 Computer terminology is used	too frequently 1 2 3 4 5 6 7 8 9 appropriately	NA
5.2.2 Terminology on the screen	ambiguous 1 2 3 4 5 6 7 8 9 precise	NA
5.3 Messages which appear on screen	inconsistent 1 2 3 4 5 6 7 8 9 consistent	NA
5.3.1 Position of instructions on the screen	inconsistent 1 2 3 4 5 6 7 8 9 consistent	NA
5.4 Messages which appear on screen	confusing 1 2 3 4 5 6 7 8 9 clear	NA
5.4.1 Instructions for commands or functions	confusing 1 2 3 4 5 6 7 8 9 clear	NA
5.4.2 Instructions for correcting errors	confusing 1 2 3 4 5 6 7 8 9 clear	NA
5.5 Computer keeps you informed about what it is doing	never 1 2 3 4 5 6 7 8 9 always	NA
5.5.1 Animated cursors keep you informed	never 1 2 3 4 5 6 7 8 9 always	NA
5.5.2 Performing an operation leads to a predictable result	never 1 2 3 4 5 6 7 8 9 always	NA
5.5.3 Controlling amount of feedback	impossible 1 2 3 4 5 6 7 8 9 easy	NA
5.5.4 Length of delay between operations	unacceptable 1 2 3 4 5 6 7 8 9 acceptable	NA
5.6 Error messages	unhelpful 1 2 3 4 5 6 7 8 9 helpful	NA
5.6.1 Error messages clarify the problem	never 1 2 3 4 5 6 7 8 9 always	NA
5.6.2 Phrasing of error messages	unpleasant 1 2 3 4 5 6 7 8 9 pleasant	NA

Please write your comments about terminology and system information here:

PART 6: Learning

6.1 Learning to operate the system	difficult 1 2 3 4 5 6 7 8 9 easy	NA
6.1.1 Getting started	difficult 1 2 3 4 5 6 7 8 9 easy	NA

Table 4.1 (continued)

6.1.2 Learning advanced features	difficult 1 2 3 4 5 6 7 8 9 easy	NA
6.1.3 Time to learn to use the system	difficult 1 2 3 4 5 6 7 8 9 easy	NA
6.2 Exploration of features by trial and error	discouraging 1 2 3 4 5 6 7 8 9 encouraging	NA
6.2.1 Exploration of features	risky 1 2 3 4 5 6 7 8 9 safe	NA
6.2.2 Discovering new features	difficult 1 2 3 4 5 6 7 8 9 easy	NA
6.3 Remembering names and use of commands	difficult 1 2 3 4 5 6 7 8 9 easy	NA
6.3.1 Remembering specific rules about entering commands	difficult 1 2 3 4 5 6 7 8 9 easy	NA
6.4 Tasks can be performed in a straightforward manner	never 1 2 3 4 5 6 7 8 9 always	NA
6.4.1 Number of steps per task	too many 1 2 3 4 5 6 7 8 9 just right	NA
6.4.2 Steps to complete a task follow a logical sequence	never 1 2 3 4 5 6 7 8 9 always	NA
6.4.3 Feedback on the completion of sequence of steps	unclear 1 2 3 4 5 6 7 8 9 clear	NA

Please write your comments about learning here:

PART 7: System Capabilities

7.1 System speed	too slow 1 2 3 4 5 6 7 8 9 fast enough	NA
7.1.1 Response time for most operations	too slow 1 2 3 4 5 6 7 8 9 fast enough	NA
7.1.2 Rate information is displayed	too slow 1 2 3 4 5 6 7 8 9 fast enough	NA
7.2 The system is reliable	never 1 2 3 4 5 6 7 8 9 always	NA
7.2.1 Operations	undependable 1 2 3 4 5 6 7 8 9 dependable	NA
7.2.2 System failures occur	frequently 1 2 3 4 5 6 7 8 9 seldom	NA
7.2.3 System warns you about potential problems	never 1 2 3 4 5 6 7 8 9 always	NA

Table 4.1 (continued)

7.3 System tends to be	noisy	quiet	
	1 2 3 4 5 6 7 8 9		NA
7.3.1 Mechanical devices such as fans, disks, and printers	noisy	quiet	
	1 2 3 4 5 6 7 8 9		NA
7.3.2 Computer generated sounds	annoying	pleasant	
	1 2 3 4 5 6 7 8 9		NA
7.4 Correcting your mistakes	difficult	easy	
	1 2 3 4 5 6 7 8 9		NA
7.4.1 Correcting typos	complex	simple	
	1 2 3 4 5 6 7 8 9		NA
7.4.2 Ability to undo operations	inadequate	adequate	
	1 2 3 4 5 6 7 8 9		NA
7.5 Ease of operation depends on your level of experience	never	always	
	1 2 3 4 5 6 7 8 9		NA
7.5.1 You can accomplish tasks knowing only a few commands	with difficulty	easily	
	1 2 3 4 5 6 7 8 9		NA
7.5.2 You can use features/shortcuts	with difficulty	easily	
	1 2 3 4 5 6 7 8 9		NA

Please write your comments about system capabilities here:

PART 8: Technical Manuals and On-line help

8.1 Technical manuals are	confusing	clear	
	1 2 3 4 5 6 7 8 9		NA
8.1.1 The terminology used in the manual	confusing	clear	
	1 2 3 4 5 6 7 8 9		NA
8.2 Information from the manual is easily understood	never	always	
	1 2 3 4 5 6 7 8 9		NA
8.2.1 Finding a solution to a problem using the manual	impossible	easy	
	1 2 3 4 5 6 7 8 9		NA
8.3 Amount of help given	inadequate	adequate	
	1 2 3 4 5 6 7 8 9		NA
8.3.1 Placement of help messages on the screen	confusing	clear	
	1 2 3 4 5 6 7 8 9		NA
8.3.2 Accessing help messages	difficult	easy	
	1 2 3 4 5 6 7 8 9		NA
8.3.3 Content of on-line help messages	confusing	clear	
	1 2 3 4 5 6 7 8 9		NA
8.3.4 Amount of help given	inadequate	adequate	
	1 2 3 4 5 6 7 8 9		NA

Table 4.1 (continued)

8.3.5	Help defines specific aspects of the system	inadequately 1 2 3 4 5 6 7 8 9 adequately	NA
8.3.6	Finding specific information using the on-line help	difficult 1 2 3 4 5 6 7 8 9 easy	NA
8.3.7	On-line help	useless 1 2 3 4 5 6 7 8 9 helpful	NA

Please write your comments about technical manuals and on-line help here:

PART 9: On-line Tutorials

9.1	Tutorial was	useless 1 2 3 4 5 6 7 8 9 helpful	NA
9.1.1	Accessing on-line tutorial	difficult 1 2 3 4 5 6 7 8 9 easy	NA
9.2	Maneuvering through the tutorial was	difficult 1 2 3 4 5 6 7 8 9 easy	NA
9.2.1	Tutorial is meaningfully structured	never 1 2 3 4 5 6 7 8 9 always	NA
9.2.2	The speed of presentation was	unacceptable 1 2 3 4 5 6 7 8 9 acceptable	NA
9.3	Tutorial content was	useless 1 2 3 4 5 6 7 8 9 helpful	NA
9.3.1	Information for specific aspects of the system were complete and informative	never 1 2 3 4 5 6 7 8 9 always	NA
9.3.2	Information was concise and to the point	never 1 2 3 4 5 6 7 8 9 always	NA
9.4	Tasks can be completed	with difficulty 1 2 3 4 5 6 7 8 9 easily	NA
9.4.1	Instructions given for completing tasks	confusing 1 2 3 4 5 6 7 8 9 clear	NA
9.4.2	Time given to perform tasks	inadequate 1 2 3 4 5 6 7 8 9 adequate	NA
9.5	Learning to operate the system using the tutorial was	difficult 1 2 3 4 5 6 7 8 9 easy	NA
9.5.1	Completing system tasks after using only the tutorial	difficult 1 2 3 4 5 6 7 8 9 easy	NA

Please write your comments about on-line tutorials here:

Table 4.1 (continued)

PART 10: Multimedia

10.1 Quality of still pictures/photographs	bad 1 2 3 4 5 6 7 8 9 good	NA
10.1.1 Pictures/Photos	fuzzy 1 2 3 4 5 6 7 8 9 clear	NA
10.1.2 Picture/Photo brightness	dim 1 2 3 4 5 6 7 8 9 bright	NA
10.2 Quality of movies	bad 1 2 3 4 5 6 7 8 9 good	NA
10.2.1 Focus of movie images	fuzzy 1 2 3 4 5 6 7 8 9 clear	NA
10.2.2 Brightness of movie images	dim 1 2 3 4 5 6 7 8 9 bright	NA
10.2.3 Movie window size is adequate	never 1 2 3 4 5 6 7 8 9 always	NA
10.3 Sound output	inaudible 1 2 3 4 5 6 7 8 9 audible	NA
10.3.1 Sound output	choppy 1 2 3 4 5 6 7 8 9 smooth	NA
10.3.2 Sound output	garbled 1 2 3 4 5 6 7 8 9 clear	NA
10.4 Colors used are	unnatural 1 2 3 4 5 6 7 8 9 natural	NA
10.4.1 Amount of colors available	inadequate 1 2 3 4 5 6 7 8 9 adequate	NA

Please write your comments about multimedia here:

PART 11: Teleconferencing

11.1 Setting up for conference	difficult 1 2 3 4 5 6 7 8 9 easy	NA
11.1.1 Time for establishing the connections to others	too long 1 2 3 4 5 6 7 8 9 just right	NA
11.1.2 Number of connections possible	too few 1 2 3 4 5 6 7 8 9 enough	NA
11.2 Arrangement of windows showing connecting groups	confusing 1 2 3 4 5 6 7 8 9 clear	NA
11.2.1 Window with view of your own group is of appropriate size	never 1 2 3 4 5 6 7 8 9 always	NA

Table 4.1 (continued)

11.2.2 Window(s) with view of connecting group(s) is of appropriate size	never	1 2 3 4 5 6 7 8 9	always NA
11.3 Determining the focus of attention during conference was	confusing	1 2 3 4 5 6 7 8 9	clear NA
11.3.1 Telling who is speaking	difficult	1 2 3 4 5 6 7 8 9	easy NA
11.4 Video image flow	choppy	1 2 3 4 5 6 7 8 9	smooth NA
11.4.1 Focus of video image	fuzzy	1 2 3 4 5 6 7 8 9	clear NA
11.5 Audio output	inaudible	1 2 3 4 5 6 7 8 9	audible NA
11.5.1 Audio is in sync with video images	never	1 2 3 4 5 6 7 8 9	always NA
11.6 Exchanging data	difficult	1 2 3 4 5 6 7 8 9	easy NA
11.6.1 Transmitting files	difficult	1 2 3 4 5 6 7 8 9	easy NA
11.6.2 Retrieving files	difficult	1 2 3 4 5 6 7 8 9	easy NA
11.6.3 Using on-line chat	difficult	1 2 3 4 5 6 7 8 9	easy NA
11.6.4 Using shared workspace	difficult	1 2 3 4 5 6 7 8 9	easy NA

Please write your comments about teleconferencing here:

PART 12: Software Installation

12.1 Speed of installation	slow	1 2 3 4 5 6 7 8 9	fast NA
12.2 Customization	difficult	1 2 3 4 5 6 7 8 9	easy NA
12.2.1 Installing only the software you want	confusing	1 2 3 4 5 6 7 8 9	clear NA
12.3 Informs you of its progress	never	1 2 3 4 5 6 7 8 9	always NA
12.4 Gives a meaningful explanation when failures occur	never	1 2 3 4 5 6 7 8 9	always NA

Please write your comments about software installation here:

An acceptance test might specify the following:

> The subjects will be 35 secretaries hired from an employment agency. They have no word-processing experience, but have typing skills in the range of 35 to 50 words per minute. They will be given 45 minutes of training on the basic features. At least 30 of the 35 secretaries should be able to complete, within 30 minutes, 80 percent of the typing and editing tasks in the enclosed benchmark test correctly.

Another testable requirement for the same system might be this:

> After four half-days of regular use of the system, 25 of these 35 secretaries should be able to carry out, within 20 minutes, the advanced editing tasks in the second benchmark test, and should make fewer than six errors.

This second acceptance test captures performance after regular use. The choice of the benchmark tests is critical and is highly system dependent. The test materials and procedures must also be refined by pilot testing before use.

A third item in the acceptance test plan might focus on retention:

> After two weeks, at least 15 of the test subjects should be recalled and should perform the third benchmark test. In 40 minutes, at least 10 of the subjects should be able to complete 75 percent of the tasks correctly.

In a large system, there may be eight or 10 such tests to carry out on different components of the interface and with different user communities. Other criteria such as subjective satisfaction, output comprehensibility, system response time, installation procedures, printed documentation, or graphics appeal may also be considered in acceptance tests of complete commercial products.

If they establish precise acceptance criteria, both the customer and the interface developer can benefit. Arguments about the user friendliness are avoided, and contractual fulfillment can be demonstrated objectively. Acceptance tests differ from usability tests in that the atmosphere may be adversarial, so outside testing organizations are often appropriate to ensure neutrality. The central goal of acceptance testing is not to detect flaws, but rather to verify adherence to requirements.

Once acceptance testing has been successful, there may be a period of field testing before national or international distribution. In addition to further refining the user interface, field tests can improve training methods, tutorial materials, telephone-help procedures, marketing methods, and publicity strategies.

The goal of early expert reviews, usability testing, surveys, acceptance testing, and field testing is to force as much as possible of the evolutionary

development into the prerelease phase, when change is relatively easy and inexpensive to accomplish.

4.6 Evaluation During Active Use

A carefully designed and thoroughly tested system is a wonderful asset, but successful active use requires constant attention from dedicated managers, user-services personnel, and maintenance staff. Everyone involved in supporting the user community can contribute to system refinements that provide ever higher levels of service. You cannot please all of the users all of the time, but earnest effort will be rewarded by the appreciation of a grateful user community. Perfection is not attainable, but percentage improvements are possible and are worth pursuing.

Gradual system dissemination is useful so that problems can be repaired with minimal disruption. As more and more people use the system, major changes should be limited to an annual or semiannual system revision that is announced adequately. If system users can anticipate the change, then resistance will be reduced, especially if they have positive expectations of improvement. More frequent changes are expected in the rapidly developing World Wide Web environment, but a balance between stable access to key resources even as novel services are added may be the winning policy.

4.6.1 Interviews and focus-group discussions

Interviews with individual users can be productive because the interviewer can pursue specific issues of concern. After a series of individual discussions, *focus-group discussions* are valuable to ascertain the universality of comments. Interviewing can be costly and time consuming, so usually only a small fraction of the user community is involved. On the other hand, direct contact with users often leads to specific, constructive suggestions.

A large corporation conducted 45-minute interviews with 66 of the 4300 users of an internal message system. The interviews revealed that the users were happy with some aspects of the functionality, such as the capacity to pick up messages at any site, the legibility of printed messages, and the convenience of after-hours access. However, the interviews also revealed that 23.6 percent of the users had concerns about reliability, 20.2 percent thought that using the system was confusing, and 18.2 percent said convenience and accessibility could be improved, whereas only 16.0 percent expressed no

concerns. Later questions in the interview explored specific features. As a result of this interview project, a set of 42 enhancements to the system was proposed and implemented. The designers of the system had earlier proposed an alternate set of enhancements, but the results of the interviews led to a changed set of priorities that more closely reflected the users' needs.

4.6.2 Continuous user-performance data logging

The software architecture should make it easy for system managers to collect data about the patterns of system usage, speed of user performance, rate of errors, or frequency of requests for online assistance. Logging data provide guidance in the acquisition of new hardware, changes in operating procedures, improvements to training, plans for system expansion, and so on.

For example, if the frequency of each error message is recorded, then the highest-frequency error is a candidate for attention. The message could be rewritten, training materials could be revised, the software could be changed to provide more specific information, or the command syntax could be simplified. Without specific logging data, the system-maintenance staff has no way of knowing which of the many hundreds of error-message situations is the biggest problem for users. Similarly, staff should examine messages that never appear, to see whether there is an error in the code or whether users are avoiding use of some facility.

If logging data are available for each command, each help screen, and each database record, then changes to the human–computer interface can be made to simplify access to frequently used features. Managers also should examine unused or rarely used facilities to understand why users are avoiding those features. Logging of the Thomas system for access to U.S. Congress legislation revealed high-frequency terms, such as *abortion, gun control,* and *balanced budget* that could be used in a browse list of hot topics (Croft et al, 1995). Logging in an educational database identified frequently used as well and rarely used paths and features (Marchionini and Crane, 1994).

A major benefit of usage-frequency data is the guidance that they provide to system maintainers in optimizing performance and in reducing costs for all participants. This latter argument may yield the clearest advantage to cost-conscious managers, whereas the increased quality of the interface is an attraction to service-oriented managers.

Logging may be well intentioned, but users' rights to privacy deserve to be protected. Links to specific user names should not be collected, unless necessary. When logging aggregate performance crosses over to monitoring individual activity, managers must inform users of what is being monitored and how the information will be used. Although organizations may have a right to ascertain worker performance, workers should be able to view the results and to discuss the implications. If monitoring is surreptitious and is later discovered, resulting worker mistrust of management could be more

damaging than the benefits of the collected data. Manager and worker cooperation to improve productivity, and worker participation in the process and benefits, are advised.

4.6.3 Online or telephone consultants

Online or *telephone consultants* can provide extremely effective and personal assistance to users who are experiencing difficulties. Many users feel reassured if they know that there is a human being to whom they can turn when problems arise. These consultants are an excellent source of information about problems users are having and can suggest improvements and potential extensions.

Many organizations offer a toll-free number via which the users can reach a knowledgeable consultant; others charge for consultation by the minute. On some network systems, the consultants can monitor the user's computer and see the same displays that the user sees while maintaining telephone voice contact. This service can be extremely reassuring: Users know that someone can walk them through the correct sequence of screens to complete their tasks.

America Online provides live (real-time) chat rooms for discussion of user problems. Users can type their questions and get responses promptly. Many groups maintain a standard electronic-mail address of staff@<organization> that allows users to get help from whomever is on duty. My several successful experiences of getting quick help late at night from our departmental staff have remained firmly in my memory. On one occasion, they helped me to unpack a file in an unfamiliar format; on another, they recovered an inadvertently deleted file.

4.6.4 Online suggestion box or trouble reporting

Electronic mail can be employed to allow users to send messages to the maintainers or designers. Such an *online suggestion box* encourages some users to make productive comments, since writing a letter may be seen as requiring too much effort.

A Library of Congress website that invites comments gets 10 to 20 per day, including thoughtful ones such as this:

> I find as I get searching through the various Web pages . . . that I am left with an unsatisfied feeling. I have been sitting in front of the PC for close to an hour . . . and have been stopped and/or slowed due to items that can be directly related to web server design.
>
> First off, the entry pages are too big and disorganized. Those links that do exist do not have adequate enough descriptions to direct a user to the information they desire. In addition, the use of a search engine would greatly facilitate sifting through the abundance of information that is thrown at the user with any one of these links. Links should be short, sweet, and specific. Large amounts of material should not be included in one document on a busy server. . . .

> Breaking up these larger documents into smaller, well organized documents may seem to create an additional burden on programming. However, if intelligence is used in the creation of such systems, it would not take much . . .

In fact, the search engine that this user wanted was available, but he could not find it, and larger documents were broken into smaller segments. A reply helped to get this user what he was seeking, and his message also led to design changes that made the interface features more visible.

An internet directory service for personal names, Knowbot Information Service, offers a `gripe` command with the invitation "Place a compliment or complaint in the KIS log file." Another service simply has a button labeled "Tell us what you think."

A large corporation installed a full-screen, fill-in-the-blanks form for user problem reports, and received 90 comments on a new internal system within three months. The user's identification number and name were entered automatically, and the user moved a cursor to indicate which subsystem was causing a problem and what the problem's seriousness was (showstopper, annoyance, improvement, other). Each problem report received a dated and signed response that was stored on a file for public reading.

4.6.5 Online bulletin board or newsgroup

Some users may have a question about the suitability of a software package for their application, or may be seeking someone who has had experience using an interface feature. They do not have any individual in mind, so electronic mail does not serve their needs. Many interface designers offer users an *electronic bulletin board* or *newsgroup* (see Section 14.3) to permit posting of open messages and questions. These newsgroups cover programming languages, software tools, or task domains. There are also mailing lists for interface designers, such as the one established on the internet by the Human Factors and Ergonomics Society's Computer Systems Technical Group (send electronic mail to listserv@listserv.vt.edu with this line: `subscribe cstg-L <your full name>`)

Some professional societies offer bulletin boards by way of networks such as America Online, Prodigy, and CompuServe. These bulletin boards may offer information services or permit downloading of software.

Bulletin-board software systems usually offer a list of item headlines, allowing users the opportunity to select items for display. New items can be added by anyone, but usually someone monitors the bulletin board to ensure that offensive, useless, or repetitious items are removed.

4.6.6 User newsletters and conferences

When there is a substantial number of users who are geographically dispersed, managers may have to work harder to create a sense of community. *Newsletters* that provide information about novel interface facilities, sugges-

tions for improved productivity, requests for assistance, case studies of successful applications, or stories about individual users can promote user satisfaction and knowledge. Printed newsletters are more traditional and have the advantage that they can be carried away from the workstation. A printed newsletter has an appealing air of respectability. Online newsletters are less expensive and more rapidly disseminated. World Wide Web or CD-ROM newsletters are appealing if collections of images are included or large datasets are anticipated.

Personal relationships established by face-to-face meetings also increase the sense of community among users. *Conferences* allow workers to exchange experiences with colleagues, promote novel approaches, stimulate greater dedication, encourage higher productivity, and develop a deeper relationship of trust. Ultimately, it is the people who matter in an organization, and human needs for social interaction should be satisfied. Every technical system is also a social system that needs to be encouraged and nurtured.

By soliciting user feedback in any of these ways, managers can gauge user attitudes and elicit useful suggestions. Furthermore, users may have more positive attitudes toward the interface if they see that the managers genuinely desire comments and suggestions.

4.7 Controlled Psychologically Oriented Experiments

Scientific and engineering progress is often stimulated by improved techniques for precise measurement. Rapid progress in the designs of interfaces will be stimulated as researchers and practitioners evolve suitable human-performance measures and techniques. We have come to expect that automobiles will have miles-per-gallon reports pasted to the window, appliances will have energy-efficiency ratings, and textbooks will be given grade-level designations; soon, we will expect software packages to show learning-time estimates and user-satisfaction indices from appropriate evaluation sources.

Academic and industrial researchers are discovering that the power of the traditional scientific method can be fruitfully employed in the study of interfaces (Barnard, 1991). They are conducting numerous experiments that are uncovering basic design principles. The outline of the scientific method as applied to human–computer interaction might include these tasks:

- Deal with a practical problem and consider the theoretical framework.
- State a lucid and testable hypothesis.
- Identify a small number of independent variables that are to be manipulated.
- Carefully choose the dependent variables that will be measured.

- Judiciously select subjects, and carefully or randomly assign subjects to groups.
- Control for biasing factors (nonrepresentative sample of subjects or selection of tasks, inconsistent testing procedures).
- Apply statistical methods to data analysis.
- Resolve the practical problem, refine the theory, and give advice to future researchers.

The classic experimental methods of psychology are being enhanced to deal with the complex cognitive tasks of human performance with information and computer systems. The transformation from Aristotelian introspection to Galilean experimentation that took two millennia in physics is being accomplished in two decades in the study of human–computer interaction.

The reductionist approach required for controlled experimentation yields narrow but reliable results. Through multiple replications with similar tasks, subjects, and experimental conditions, reliability and validity can be enhanced. Each small experimental result acts like a tile in the mosaic of human performance with computer-based information systems.

Managers of actively used systems are also coming to recognize the power of controlled experiments in fine tuning the human–computer interface. As proposals are made for new menu structures, novel cursor-control devices, and reorganized display formats, a carefully controlled experiment can provide data to support a management decision. Fractions of the user population could be given proposed improvements for a limited time, and then performance could be compared with the control group. Dependent measures could include performance times, user-subjective satisfaction, error rates, and user retention over time.

Experimental design and statistical analysis are complex topics (Hays, 1988; Cozby, 1996; Runyon and Haber, 1996; Winer et al., 1991.) Novice experimenters would be well advised to collaborate with experienced social scientists and statisticians.

4.8 Practitioner's Summary

Interface developers evaluate their designs by conducting expert reviews, usability tests, surveys, and rigorous acceptance tests. Once systems are released, developers perform continuous performance evaluations by interviews or surveys, and by logging user performance in a way that respects the privacy of users. If you are not measuring, you are not doing human factors!

Successful system managers understand that they must work hard to establish a relationship of trust with the user community. In addition to pro-

viding a properly functioning system, computer service managers and information-systems directors recognize the need to create social mechanisms for feedback, such as online surveys, interviews, discussions, consultants, suggestion boxes, bulletin boards, newsletters, and conferences.

4.9 Researcher's Agenda

Researchers can contribute their experience with experimentation to developing techniques for system evaluation. Guidance in conducting pilot studies, acceptance tests, surveys, interviews, and discussions would benefit commercial development groups. Experts in constructing psychological tests would be extremely helpful in preparing a validated and reliable test instrument for subjective evaluation of interactive systems. Such a standardized test would allow independent groups to compare the acceptability of their systems. In addition, assessment methods for user skill levels with software would be helpful in job-placement and training programs.

Clinical psychologists, psychotherapists, and social workers could contribute to training online or as telephone consultants—after all, helping troubled users is a human-relationship issue. Finally, more input from experimental, cognitive, and clinical psychologists would help computer specialists to recognize the importance of the human aspects of computer use. What techniques can reduce novice user anxiety? How can life-critical applications for experienced professionals be tested reliably?

World Wide Web Resources WWW

Prototyping and usability testing methods are covered with some information on evaluation methods, such as surveys. The full text of our QUIS is available online.

http://www.aw.com/DTUI

References

Barnard, Phil, The contributions of applied cognitive psychology to the study of human–computer interaction. In Shackel, B. and Richardson, S. (Editors), *Human Factors for Informatics Usability*, Cambridge University Press, Cambridge, U.K. (1991), 151–182.

Chin, John P., Diehl, Virginia A., and Norman, Kent L., Development of an instrument measuring user satisfaction of the human–computer interface, *Proc. CHI '88—Human Factors in Computing Systems*, ACM, New York (1988), 213–218.

Coleman, William D. and Williges, Robert C., Collecting detailed user evaluations of software interfaces, *Proc. Human Factors Society—Twenty-Ninth Annual Meeting*, Santa Monica, CA (1985), 204–244.

Cozby, Paul C., *Methods in Behavioral Research* (Sixth Edition), Mayfield, Mountain View, CA (1996).

Croft, W. Bruce, Cook, Robert, and Wilder, Dean, Providing government information on the internet: Experiences with THOMAS, *Proc. Digital Libraries '95 Conference*, ACM, New York (1995). Also available at http://www.csdl.tamu.edu/DL95/papers/croft/croft.html

Curtis, Bill, Defining a place for interface engineering, *IEEE Software*, 9, 2 (March 1992), 84–86.

Dumas, Joseph and Redish, Janice, *A Practical Guide to Usability Testing*, Ablex, Norwood, NJ (1993).

Gould, John, How to design usable systems. In Helander, Martin (Editor), *Handbook of Human–Computer Interaction*, North-Holland, Amsterdam, The Netherlands (1988), 757–789.

Gould, John D., Boies, Stephen J., and Lewis, Clayton, Making usable, useful productivity-enhancing computer applications, *Communications of the ACM*, 34, 1 (January 1991), 75–85.

Harrison, Beverly L., Video annotation and multimedia interfaces: From theory to practice, *Proc. Human Factors Society Thirty-Fifth Annual Meeting* (1991), 319–322.

Hays, William L., *Statistics* (Fourth Edition), Holt, Rinehart and Winston, New York (1988).

Hix, Deborah and Hartson, H. Rex, *Developing User Interfaces: Ensuring Usability Through Product and Process*, John Wiley and Sons, New York (1993).

Jeffries, R., Miller, J. R., Wharton, C., and Uyeda, K. M., User interface evaluation in the real world: A comparison of four techniques, *Proc. ACM CHI91 Conf.* (1991), 119–124.

Karat, Claire-Marie, A business case approach to usability. In Bias, Randolph, and Mayhew, Deborah (Editors), *Cost-Justifying Usability*, Academic Press, New York (1994), 45–70.

Karat, Claire-Marie, Campbell, Robert, and Fiegel, T., Comparison of empirical testing and walkthrough methods in user interface evaluation, *Proc. CHI '92—Human Factors in Computing Systems*, ACM, New York (1992), 397–404.

Kirakowski, J. and Corbett, M. SUMI: The Software Usability Measurement Inventory, *British Journal of Educational Technology*, 24, 3 (1993), 210–212.

Landauer, Thomas K., *The Trouble with Computers: Usefulness, Usability, and Productivity*, MIT Press, Cambridge, MA (1995).

Lewis, James R., IBM computer usability satisfaction questionnaires: Psychometric evaluation and instructions for use, *International Journal of Human–Computer Interaction*, 7, 1 (1995), 57–78.

Lund, Michelle A., Evaluating the user interfaces: The candid camera approach, *Proc. CHI '85—Human Factors in Computing Systems*, ACM, New York (1985), 93–97.

Marchionini, Gary and Crane, Gregory, Evaluating hypermedia and learning: Methods and results from the Perseus Project, *ACM Transactions on Information Systems*, 12, 1 (1994), 5–34.

Newman, William M. and Lamming, Michael G., *Interactive System Design*, Addison-Wesley, Reading, MA (1995).

Nielsen, Jakob (Editor), Special Issue on Usability Laboratories, *Behaviour & Information Technology*, 13, 1 & 2 (January–April 1994).

Nielsen, Jakob, *Usability Engineering*, Academic Press, New York (1993).

Nielsen, Jakob and Mack, Robert (Editors), *Usability Inspection Methods*, John Wiley and Sons, New York (1994).

Oppenheim, Abraham N., *Questionnaire Design, Interviewing, and Attitude Measurement*, Pinter Publishers, New York (1992).

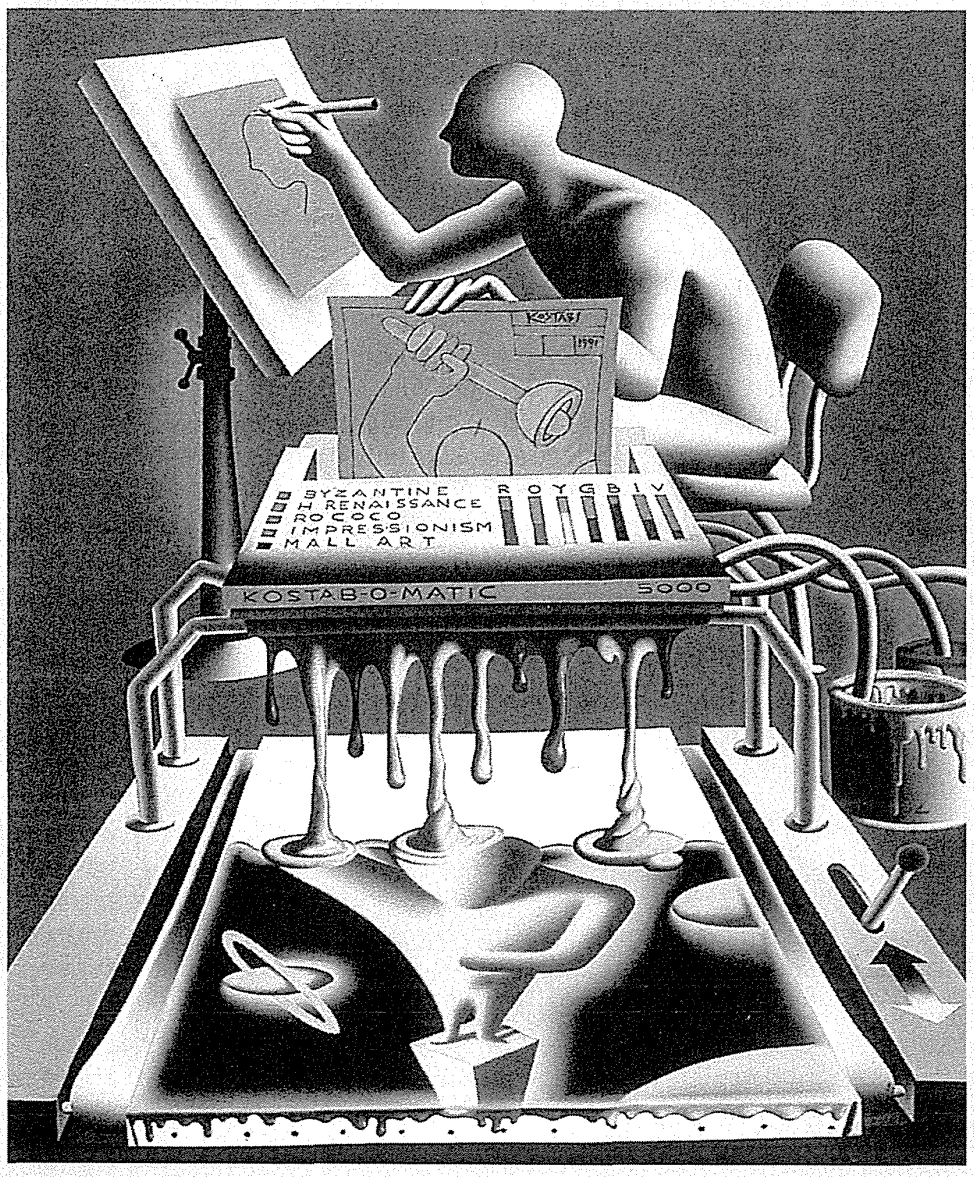
Preece, Jenny, Rogers, Yvonne, Sharp, Helen, Benyon, David, Holland, Simon, and Carey, Tom, *Human–Computer Interaction*, Addison-Wesley, Reading, MA (1994).

Runyon, Richard P. and Haber, Audrey, *Fundamentals of Behavioral Statistics* (Eighth Edition), McGraw-Hill, New York (1996).

Wharton, Cathleen, Rieman, John, Lewis, Clayton, and Polson, Peter, The cognitive walkthrough method: A practitioner's guide. In Nielsen, Jakob and Mack, Robert (Editors), *Usability Inspection Methods*, John Wiley and Sons, New York (1994).

Winer, B. J., Brown, Donald R., and Michels, Kenneth M., *Statistical Principles in Experimental Design*, McGraw-Hill, New York (1991).

Yourdon, Edward, *Structured Walkthroughs* (Fourth Edition), Yourdon Press, Englewood Cliffs, NJ (1989).



Mark Kostabi, *Automatic Painting*, 1991

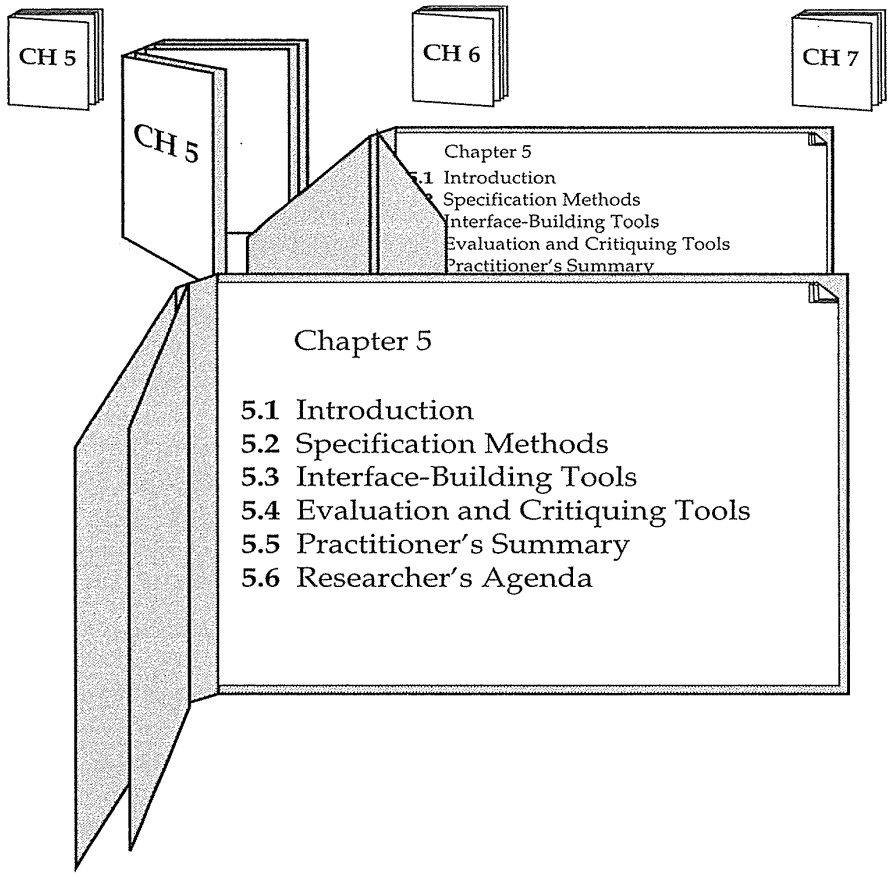
CHAPTER

5

Software Tools

There is great satisfaction in building good tools for other people to use.

Freeman Dyson, ***Disturbing the Universe,*** **1979**



5.1 Introduction

Log cabins were often built by settlers for personal housing on the American frontier, just as early user interfaces were built by programmers for their own use. As housing needs changed, windows and rooms were added in a process of iterative refinement, and dirt floors gave way to finished wood. Log cabins are still being built according to personal taste by rugged individualists, but modern private homes, apartment buildings, schools, hospitals, and offices require specialist training, careful planning, and special equipment.

The emergence of user-interface architects, design and specification methods, standard components, and automated tools for construction are indicators of the maturation of our field. There will always be room for the innovator and the eccentric, but the demands of modern life require user-

interface architects to build reliable, standard, safe, inexpensive, effective, and widely acceptable user interfaces on a predictable schedule (Carey, 1988).

Building and user-interface architects must have simple and quick methods of sketching to give their clients a way to identify needs and preferences. Then, they need precise methods for working out the details with the clients (detailed floorplans become transition diagrams, screen layouts, and menu trees), for coordinating with specialized colleagues (plumbers and electricians become graphic designers and technical writers), and for telling the builders (or software engineers) what to do.

Like building architects, successful user-interface architects know that it makes good sense to complete the design before they start building, even though they know that, in the process of construction, some changes will have to be made. With large projects, multiple designers (structural engineers for the steel framework, interior designers for space planning, and decorators for the esthetics) will be necessary. The size and importance of each project will determine the level of design effort and the number of participants. Just as there are specialists for airports, hospitals, and schools, there are user-interfaces specialists for air-traffic–control, medical, and educational applications.

This chapter begins with user-interface specification methods, moves to software tools to support design and software engineering, and then closes with evaluation and critiquing tools. These tools are increasingly graphical in their user interfaces, enabling designers and programmers to build interfaces rapidly by dragging components and linking functions together. User-interface building tools have matured rapidly in the past few years, and have radically changed the nature of software development. Productivity gains of 50 to 500 percent above previous methods have been documented for many standard GUIs. But, even as the power tools for established styles improve and gain acceptance, programmers will always have to handcraft novel interface styles.

5.2 Specification Methods

The first asset in making designs is a good notation to record and discuss alternate possibilities. The default language for specifications in any field is the designer's natural language, such as English, and a sketchpad or blackboard. But *natural-language specifications* tend to be lengthy, vague, and ambiguous, and therefore often are difficult to prove correct, consistent, or complete. *Formal* and *semiformal languages* have proved their value in many areas, including mathematics, physics, circuit design, music, and even knitting. Formal languages have a specified grammar, and effective procedures exist to determine whether a string adheres to the language's grammar.

Grammars for command languages are effective, but for GUIs the amount of syntax is small. In GUIs, a grammar might be used to describe sequences of actions, but these grammars tend to be short, making transition diagrams and graphical specifications more appealing.

Menu-tree structures are popular, and therefore specifying menu trees by simply drawing the tree and showing the menu layouts deserves attention. The more general method of *transition diagrams* has wide applicability in user-interface design. Improvements such as *statecharts* have features that are attuned to the needs of interactive systems and for widget specification. New approaches such as the *user action notation* (UAN) (Hartson et al., 1990; Chase et al., 1994) are helpful in characterizing user behavior and some aspects of system responses.

5.2.1 Grammars

In computer programming, *Backus–Naur form* (BNF) also called (*Backus normal form*) is often used to describe programming languages. High-level components are described by nonterminals, and specific strings are terminals. Let us use the example of a telephone-book entry. The nonterminals describe a person's name (composed of a last name followed by a comma and a first name) and a telephone number (composed of an area code, exchange, and local number). Names consist of strings of characters. The telephone number has three components: a three-digit area code, a three-digit exchange, and a four-digit local number.

```
<Telephone book entry> ::= <Name> <Telephone number>
<Name> ::= <Last name>, <First name>
<Last name> ::= <string>
<First name> ::= <string>
<string> ::= <character>|<character><string>
<character> ::=
   A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
<Telephone number> ::= (<area code>) <exchange>-<local number>
<area code> ::= <digit><digit><digit>
<exchange> ::= <digit><digit><digit>
<local number> ::= <digit><digit><digit><digit>
<digit> ::= 0|1|2|3|4|5|6|7|8|9
```

The left-hand side of each specification line is a nonterminal (within angle brackets) that is defined by the right-hand side. Vertical bars indicate alternatives for nonterminals and terminals. Acceptable-telephone-book entries include the following:

```
WASHINGTON, GEORGE (301) 555-1234
BEEF, STU (726) 768-7878
A, Z (999) 111-1111
```

BNF notation is used widely, even though it is incomplete and must be supplemented by ad hoc techniques for specifying the semantics, such as permissible names or area codes. The benefits are that some aspects can be written down precisely, and that software tools can be employed to verify some aspects of completeness and correctness of the grammar and of strings in the language. On the other hand, grammars are difficult to follow as they grow and are confusing for many users.

Command languages are nicely specified by BNF-like grammars, such as the task–action grammar (Section 2.2.4). Reisner (1981) expanded the idea of BNF to sequences of actions, such as pushing a button, selecting a color, or drawing a shape.

Variant forms of BNF have been created to accommodate specific situations. For example, the Unix command for copying files or directories is summarized by this extract from the online manual:

```
cp [ -ip ] filename1 filename2
cp -rR [ -ip ] directory1 directory2
cp [ -iprR ] filename ... directory
```

where the square brackets indicate that zero or more options can be included, and the −rR indicates that one of these options for recursive copying is required for copying directories.

To accommodate the richness of interactive software, *multiparty grammars* (Shneiderman, 1982) have nonterminals that are labeled by the party that produces the string (typically the user, U, or the computer, C). Nonterminals acquire values during parsing for use by other parties, and therefore error-handling rules can be included easily. This grammar describes the opening steps in a login process:

```
<Session> ::= <U: Opening> <C: Responding>
<U: Opening> ::= LOGIN <U: Name>
<U: Name> ::= <U: string>
<C: Responding> ::= HELLO [<U: Name>]
```

Here, square brackets indicate that the value of the user's name should be produced by the computer in responding to the login command.

Multiparty grammars are effective for text-oriented command sequences that have repeated exchanges, such as a bank terminal. Unfortunately, two-dimensional styles, such as form fillin or direct manipulation and graphical layouts, are more difficult to describe with multiparty grammars. Menu selection can be described by multiparty grammars, but the central aspect of tree structure and traversal is not shown conveniently in a grammar-based approach.

5.2.2 Menu-selection and dialog-box trees

For many applications a *menu-selection tree* is an excellent selection style because of the simple structure that guides designers and users alike. Guidelines for the contents of the menu trees are covered in Chapter 7. Specification methods include online tools to help in the construction of menu trees and simple drawing tools that enable designers and users to see the entire tree at one time.

Menu trees are powerful as a specification tool since they show users, managers, implementers, and other interested parties the complete and detailed coverage of the system. Like any map, a menu tree shows high-level relationships and low-level details. With large systems, the menu tree may have to be laid out on a large wall or floor, but it is important to be able to see the entire structure at once to check for consistency, completeness, and lack of ambiguity or redundancy.

Similar comments apply for dialog boxes. Printing out the dialog boxes and showing their relationships by mounting them on a wall is enormously helpful in gaining an overview of the entire system to check for consistency and completeness.

5.2.3 Transition diagrams

Menu trees are incomplete because they do not show the entire structure of possible user actions, such as returns to the previous menu, jumps to the starting menu, or detours to error handling or to help screens. However, adding all these transitions would clutter the clean structure of a menu tree. For some aspects of the design process, more precise specification of every possible transition is required. Also, for many nonmenu interaction styles, there is a set of possible states and permissible transitions among the states that may not form a tree structure. For these and other circumstances, a more general design notation known as *transition diagrams* has been used widely.

Typically, a transition diagram has a set of *nodes* that represents system states and a set of *links* between the nodes that represents possible transitions. Each link is labeled with the user action that selects that link and possible computer responses. The simple transition diagram in Fig. 5.1 (Wasserman and Shewmake, 1985) represents a numbered menu-selection system for restaurant reviews that shows what happens when the user selects numbered choices: 1 (add a restaurant to the list), 2 (provide a review of a restaurant), 3 (read a review), 4 (get help, also accessed by a ?), 5 (quit), or any other character (error message). Figure 5.2 shows its text form. Figure 5.3 shows another form of transition diagram that displays frequencies along the links.

Many forms of transition diagrams have been created with special notations to fit needs of application areas, such as air-traffic control or word pro-

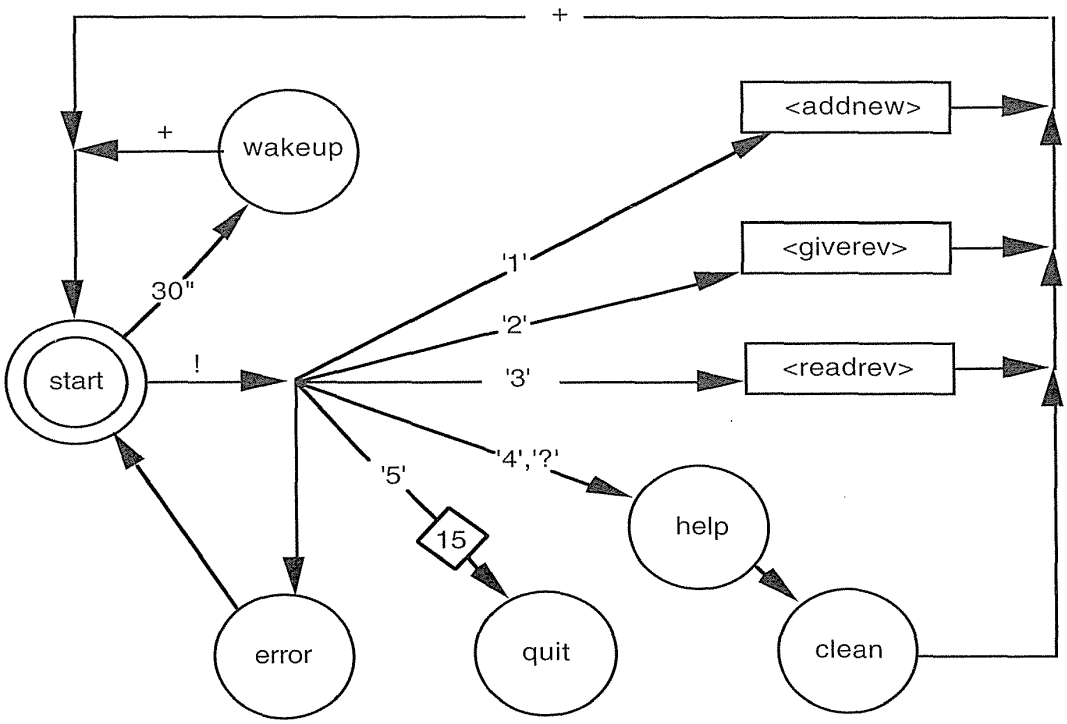


Figure 5.1

Transition diagram for a simple menu system. (Wasserman and Shewmake, 1985.)

cessing. Tools for creating and maintaining transition diagrams, dataflow diagrams, and other graphical displays are part of most *computer-assisted software engineering (CASE)* environments, such as the Software Through Pictures (Interactive Development Environments, Inc., http://www.ide.com). In most systems, the diagram is created by direct-manipulation actions, but designers can get a textual output of the transition diagram as well.

Unfortunately, transition diagrams get unwieldy as system complexity grows, and too many transitions can lead to complex spaghetti-like displays. Improvements are to replace a state transition node with a screen print to give readers a better sense of movement through the displays and dialog boxes. Such overviews are helpful in design and in training.

Designs for interfaces with hundreds of dialog boxes, or for websites with hundreds of screens, are easier to study when hung on the wall. In a memorable encounter, 350 screens of a satellite-control system were pasted on three walls of a conference room, quickly revealing the disparate styles of the design teams of the six modules. Compressed overview diagrams may be squeezed onto a single sheet of paper for user manuals, or printed as a poster to hang on users' walls.

```
node start
        cs, r2, rv, c_' Interactive Restaurant Guide', sv,
        r6, c5,  'Please make a choice:  ',
        r+2, c10,  '1:  Add new restaurant to database',
        r+2, c10,  '2:  Give review of a restaurant  ',
        r+2, c10,  '3:  Read reviews for a given restaurant',
        r+2, c10,  '4:  Help',  r+2, c10,  '5:  Quit',  r+3,c5,  'Your choice:  ',  mark_A

node help
        cs, r5, c0,  'This program stores and retrieves information on',
        r+1,  c0, 'restaurants,  with emphasis on San Francisco.',
        r+1,  c0,  'You can add or update information about restaurants',
        r+1,  c0,  'already in the database,  or obtain information about',
        r+1,  c0,  'restaurants,  including the reviews of others.',
        r+2,  c0,  'To continue, type RETURN.'

node error
        r$-1, rv, 'Illegal command.',  sv,  'Please type a number from 1 to 5.',
        r$,  'Press RETURN to continue.'
node clean
        r$-1, c1,r$,c1
node wakeup
        r$,c1,rv,'Please make a choice',sv,  tomark_A
node quit
        cs,  'Thank you very much.  Please try this program again',
        n1,'and continue to add information on restaurants.'
arc start single_key
        on  '1'  to  <addnew>
        on  '2'  to  <giverev>
        on  '3'  to  <readrev>
        on  '4',  '?'  to help
        on  '5'  to quit
        alarm 30 to wakeup
        else to error
arc error
        else to start
arc help
        skip to clean
arc clean
        else to start
arc <addnew>
        skip to start
arc <readrev>
        skip to start
arc <giverev>
        skip to start
```

Figure 5.2

Text form of Fig. 5.1. Additional information is provided by the comment lines.

5.2.4 Statecharts

Although transition diagrams are effective for following flow or action and for keeping track of the current state plus current options, they can rapidly become large and confusing. Modularity is possible if nodes are included with subgraphs, but this strategy works well with only orderly, one-in, one-out graphs. Transition diagrams also become confusing when each node must show links to a help state, jumps back to the previous or start state, and a quit

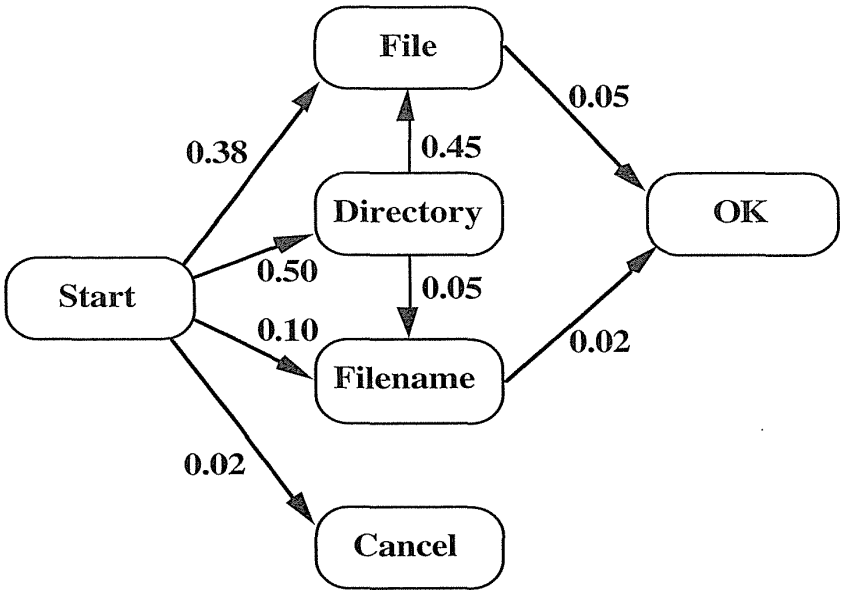


Figure 5.3

Sample transition diagram for file-manipulation actions. Link labels indicate how frequently each transition is made.

state. Concurrency and synchronization are poorly represented by transition diagrams, although some variations such as petri-nets can help. An appealing alternative is *statecharts* (Harel, 1988), which have several virtues in specifying interfaces. Because a grouping feature is offered through nested roundtangles (Fig. 5.4), repeated transitions can be factored out to the surrounding roundtangle. Extensions to statecharts—such as concurrency, external interrupt events, and user actions—are represented in Statemaster, which is a user-interface tool based on statecharts (Wellner, 1989).

Statecharts can also be extended with dataflow and constraint specification, plus embedded screen prints to show the visual states of graphical widgets (Carr, 1994). For example, in the simple case of a secure toggle switch, there are five states, so showing the visual feedback on the statechart with user-action notation (see Section 5.2.5) on the arcs helps readers to understand what is happening (Fig. 5.5).

5.2.5 User-action notation (UAN)

The grammar or diagram approaches to specification are suited for menus, commands, or form fillin, but they are clumsy with direct-manipulation interfaces, because they cannot cope conveniently with the variety of permissible

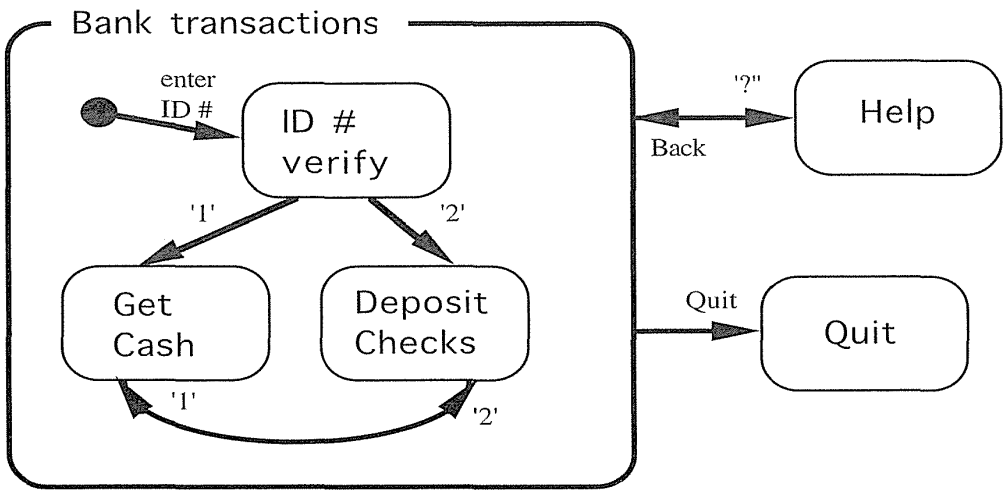


Figure 5.4

Statechart of a simplified bank transaction system showing grouping of states.

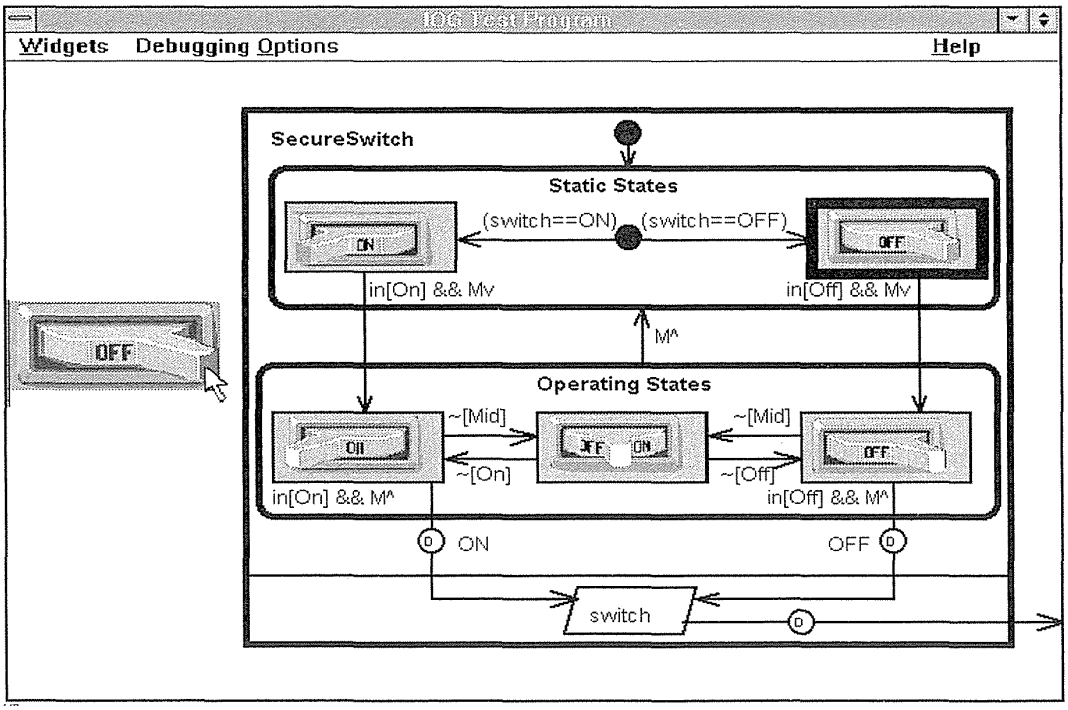


Figure 5.5

Interaction-object graphs extend statecharts with dataflow features and the user-action notation. This example shows a secure switch with bitmaps of the states at each node. (Carr, 1994)

actions and visual feedback that the system provides. In addition, direct-manipulation interfaces depend heavily on context to determine the meaning of an input. For example, a mouse-button click can mean select a file, open a window, or start an application, depending on where the cursor is when the click is applied. Similarly, it is difficult to characterize the results of dragging an icon, since they will depend on where the icon is dropped.

To cope with the rich world of direct-manipulation interfaces, high-level notations that focus on the users' tasks, that deal with pointing, dragging, and clicking, and that describe the interface feedback are more likely to be helpful. For example, to select an icon, the user must move the cursor to the icon location and click and release on the mouse button. The movement to an icon is represented by a `~[icon]` and the mouse-button motion is represented by Mv (mouse-button depress) followed by M^ (mouse-button release). The system response, which is to highlight the icon, is represented by `icon!` The sequencing is shown by a complete *user-action notation (UAN)* description (Hartson et al., 1990; Hix and Hartson, 1993):

TASK: Select an icon

User Actions	Interface Feedback
`~[icon] Mv`	`icon!`
`M^`	

A more complex task might be to delete a file; that task requires user actions of dragging a file icon around the display to a trash icon while holding down the mouse button. The interface feedback is to highlight the file that is selected and to dehighlight (`file-!` indicates dehighlight the file) other files, then to drag an outline of the file icon to the trash icon (`outline(file) > ~` means that the outline is dragged by the cursor). Then, the user drops the file-icon outline on the trash icon, the file icon is erased, and the trash icon blinks. The selected file is shown in the interface-state column:

TASK: Select an icon

User Actions	Interface Feedback	Interface State
`~[file] Mv`	`file!, forall(file!): file-!`	`selected = file`
`~[x,y]*`	`outline(file) > ~`	
`~[trash]`	`outline(file) > ~, trash!`	
`M^`	`erase(file), trash!!`	`selected = null`

The UAN has interface-specific symbols for actions (such as moving the cursor, pressing a button, entering a string, or setting a value), and for concurrency, interrupts, and feedback (such as highlighting, blinking, dragging,

rubberbanding, and erasing). The symbols were chosen to mimic the actions—such as v for button depress, ^ for button release, and ~ for cursor movement—but it still takes time to get used to this novel notation. Also, UAN does not conveniently specify rich graphics, such as drawing programs or animations, relationships across tasks, and interrupt behavior. Nonetheless, UAN is a compact, powerful, and high-level approach to specifying system behavior and describing user actions (Chase et al., 1994).

5.3 Interface-Building Tools

Specification methods are important for the design of components of a system such as command languages, data-entry sequences, and widgets. Screen-transition diagrams drawn or printed on paper are an excellent means to provide an overview of the system. They allow user-interface architects, designers, managers, users, and software engineers to sit around a table, discuss the design, and prepare for the big job that lies ahead. Paper-based designs are a great way to start, but the detailed specification of complete user interfaces requires software tools.

The good news is that there has been a rapid and remarkable proliferation of software tools to accommodate most designers and software engineers in accomplishing many design goals. These tools come in colorful shrink-wrapped boxes that emphasize convenient and rapid building of onscreen prototypes. They generally allow visual editing, so designers can immediately assess the "look" of the system and can easily change color, fonts, and layout. These direct-manipulation design tools have enabled large numbers of task-domain experts who have only modest technical training to become user-interface designers.

Other tools are powerful programming languages that include extensive toolkits that enable experienced software engineers to build a richer variety of features, but that often require twice or 20 times as much code and work. Of course, there will always be special designs that require programming in languages, such as C or C++, or even in assembly language to deal with precise timing or special hardware features.

The terminology for products varies depending on the vendor. Popular terms include Rapid Prototyper, User Interface Builder, User Interface Management System, User Interface Development Environment, Rapid Application Developer. A key distinction is how extensively the system uses convenient visual programming, a relatively simple scripting language (event or object oriented), or a more powerful general-purpose programming language.

Use of these software tools brings great benefits (Box 5.1), and is spreading widely, even as the tools are rapidly improved in successive versions.

Box 5.1

Features of user-interface–building tools.

User-interface independence

- Separate interface design from internals
- Enable multiple user-interface strategies
- Enable multiple-platform support
- Establish role of user-interface architect
- Enforce standards

Methodology and notation

- Develop design procedures
- Find ways to talk about design
- Create project management

Rapid prototyping

- Try out ideas very early
- Test, revise, test, revise, . . .
- Engage end users, managers, and customers

Software support

- Increase productivity
- Offer constraint and consistency checks
- Facilitate team approaches
- Ease maintenance

The central advantage stems from the notion of *user-interface independence*—decoupling of the user-interface design from the complexities of programming. This decoupling allows the designers to lay out sequences of displays in just a few hours, to make revisions in minutes, and to support the expert-review and usability-testing processes. The programming needed to complete the underlying system can be applied once the user-interface design has been stabilized. The user-interface prototypes can serve as specifications from which writers create user manuals, and from which software engineers build the system using other tools. The latter are required to produce a system that works just like the prototype. In fact, prototypes can be the specification in government or commercial contracts for novel software.

Some early tools were limited to doing prototyping only, but most modern tools allow for quick prototyping and then system development. The design tools enable construction of complete systems but they may run slowly, limit

the database size, or restrict users in many ways. The software-engineering tools allow construction of more robust systems, but the complexity, cost, and development time are usually greater.

An important consideration in choosing tools is whether they support cross-platform development, a strategy in which the interface can run on multiple environments such as Windows or Unix. There is a great benefit if only one program needs to be written and maintained, but the product is available on multiple platforms.

Another important consideration is whether the application allows the user interface to run under a web browser such as Netscape Navigator or Microsoft Internet Explorer. Since these browsers are written for multiple platforms, the cross-platform goal is automatically met. The World Wide Web is such a powerful force that web-oriented tools are likely to have the brightest future.

5.3.1 Design tools

User-interface architects recognize that creating quick sketches is important during the early stages of design to explore multiple alternatives, to allow communication within the design team, and to convey to clients what the product will look like. User-interface mockups can be created with paper and pencil, word processors, or slide-show presentation software (Adobe Persuasion or Microsoft PowerPoint). Resourceful designers have also built user-interface prototypes with computer-assisted–instruction software, such as Authorware, IconAuthor, or Quest, and with multimedia construction tools, such as Apple Hypercard, MacroMind Director, or Asymetrix Toolbook.

In the simplest case, designers create a slide show of still images, which are switched at a user-controlled pace. Most tools support more complete prototyping that allows users to select from menus, click on buttons, use scrolling lists, and even drag icons. Users can navigate through screens and go back to previous screens. The prototype may not have a full database, help, or other facilities, but it offers a carefully chosen path that gives a realistic presentation of what the interface will do.

Visual editing tools usually permit designers to lay out displays with cursor movements or mouse clicks, and to mark regions for selection, highlighting, or data entry. Then, designers can specify which button selection is linked to a related display or dialog box. Prototypes are excellent aids to design discussions and are effective in winning contracts, because clients can be given a rough idea of what the finished system will be like.

The early success Apple's HyperCard stimulated many competitors. These systems combine visual editing—by allowing designers to include buttons and other fields—with simple interface actions provided automatically (for example, clicking on a back-arrow would take the user to the previ-

ous card). For more complex actions, the innovative HyperTalk scripting language enables many users to create useful interfaces with only moderate training. Designers can write programs with easy-to-understand terms:

```
on mouseUp
    play "boing"
    wait for 3 seconds
    visual effect wipe left very fast to black
    click at 150,100
    type "goodbye"
end mouseUp
```

Of course, programming in such languages can become complex as the number of short code segments grows and their interrelationships become difficult to fathom.

Visual programming tools with direct manipulation, such as Prograph (Pictorius Systems), are an intriguing alternative. Prograph allows users to edit, execute, debug, and make changes during execution, with flowchart-like visual-programming tools that emphasize dataflow and have a deeply nested modular structure (Fig. 5.6). Visual programming for laboratory instruments was the motivating influence for LabVIEW (National Instruments) (Fig. 5.7), which has a flat structure of function boxes (arithmetic, Boolean, and more) linked with wires (Green and Petre, 1996).

Contemporary visual development tools such as Microsoft Visual Basic (Fig. 5.8), Borland Delphi (Fig. 5.9), and Symantec Cafe (Fig. 5.10) have easy-to-use design tools for dragging buttons, labels, data-entry fields, combo boxes, and more onto a workspace to assemble the visual interface. Then, users write code in a scripting language that is an extension of Basic, object-oriented Pascal, or Java to implement the actions. The visual editors in these products reduce design time for user interfaces dramatically, if designers are content to use the supplied widgets, such as labels, data-entry boxes, scroll bars, scrolling lists, or text-entry areas. Adding new widgets takes programming skill, but there are large libraries of widgets for sale. Delphi's compiled Pascal code runs faster than the interpreted Basic, and Delphi also provides good support for database access, but newer versions of each product are likely to challenge each other.

5.3.2 Software-engineering tools

Experienced programmers often build user interfaces with general-purpose programming languages such as C or C++, but this approach is giving way to using facilities that are especially tuned to user-interface development and web access (Olsen, 1991; Myers, 1995).

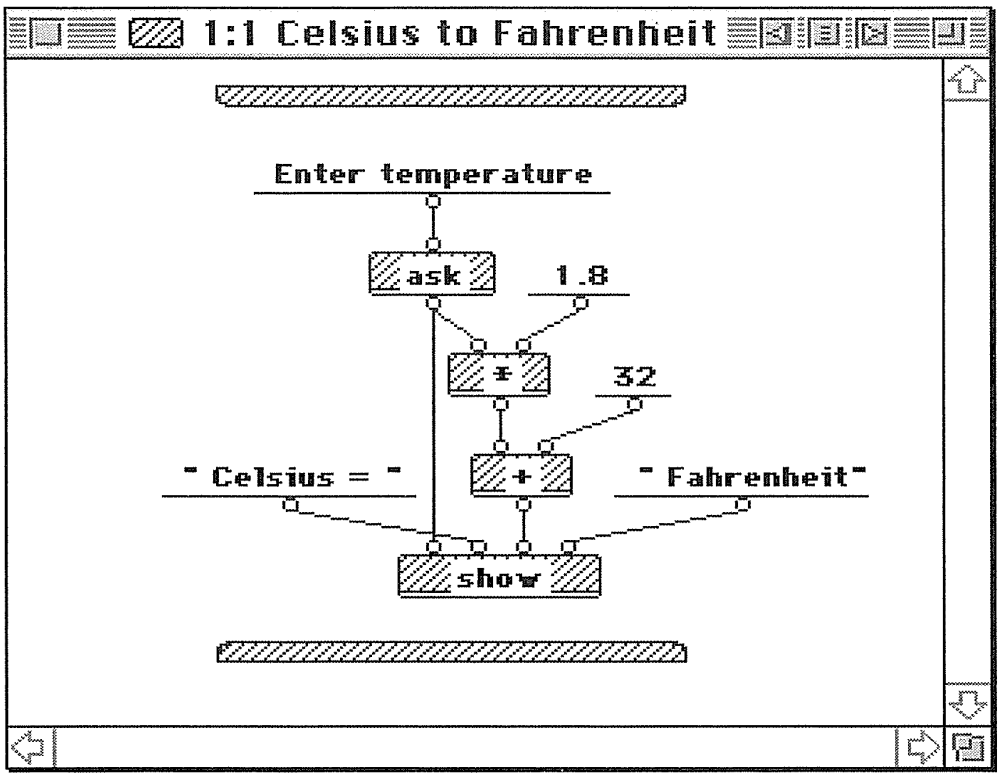


Figure 5.6

Prograph CPX, a visual language that uses object-oriented programming techniques, including inheritance, encapsulation, and polymorphism. This simple example shows a common programming problem. (Used with permission of Pictorius Inc., Halifax, Nova Scotia, Canada.)

Some products provide user-interface program libraries, often called *toolkits*, that offer common widgets, such as windows, scroll bars, pull-down or pop-up menus, data-entry fields, buttons, and dialog boxes. Programming languages with accompanying libraries are familiar to experienced programmers and afford great flexibility. However, toolkits can become complex, and the programming environments for those, such as Microsoft Windows Developer's Toolkit, Apple Macintosh MacApp, and Unix X-Windows toolkit (Xtk), require months of learning for programmers to gain proficiency. Even then, the burden in creating applications is great, and maintenance is difficult. The advantage is that the programmer has extensive control and great flexibility in creating the interface. Toolkits have become popular with programmers, but they provide only partial support for consis-

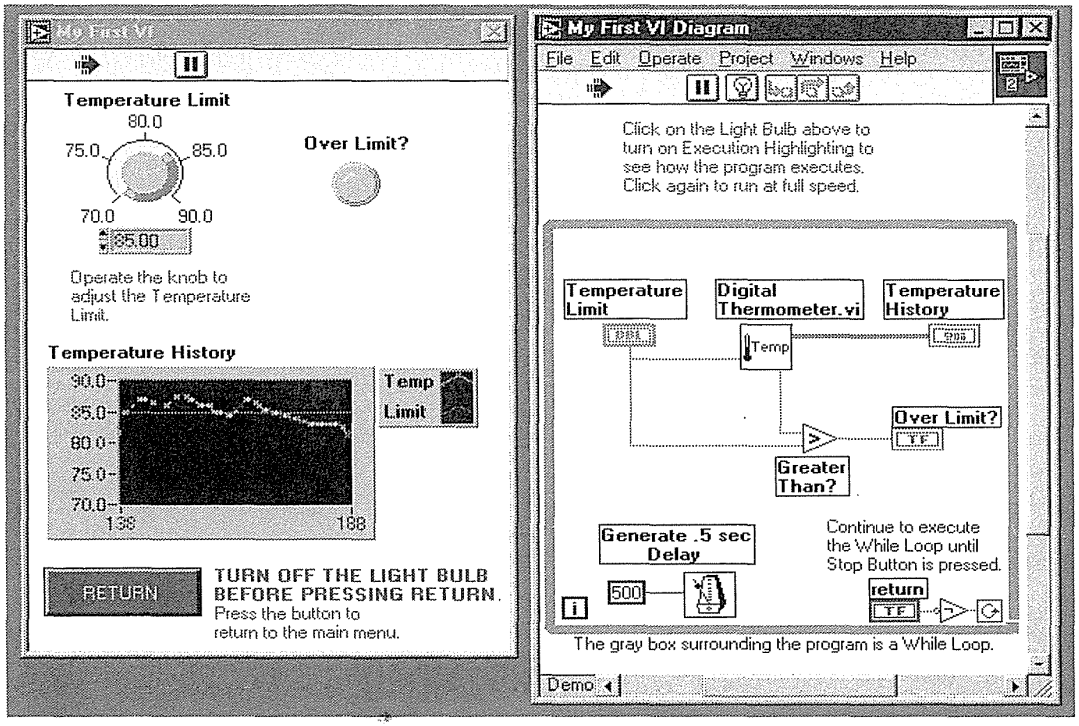


Figure 5.7

LabVIEW enables users to develop virtual instruments in a visual-programming environment. In this simple demo program, the virtual instrument on the left is controlled by the program on the right, which can show an animation of its execution. (Reprinted with permission of copyright owner, National Instruments Corporation (Austin,TX). LabVIEW is a registered trademark of National Instruments.)

tency, and designers and managers must still depend heavily on experienced programmers. The Motif example in Fig. 5.11 conveys the challenge of programming user interfaces in X.

To lighten the burden of programming, Ousterhout developed a simpler scripting language called Tcl and an accompanying toolkit called Tk (Ousterhout, 1994). Their great success was due to the relative ease of use of Tcl and the useful widgets in Tk, such as the text and canvas. Tcl is interpreted, so development is rapid, and its cross-platform capabilities are further attractions. The absence of a visual editor discourages some users, but Tcl's convenience in gluing together components has overcome the objections of most critics. This sample menu-construction program illustrates Tcl scripting (Martland, 1994, http://http2.brunel.ac.uk:8080/~csstddm/TCL2/TCL2.html):

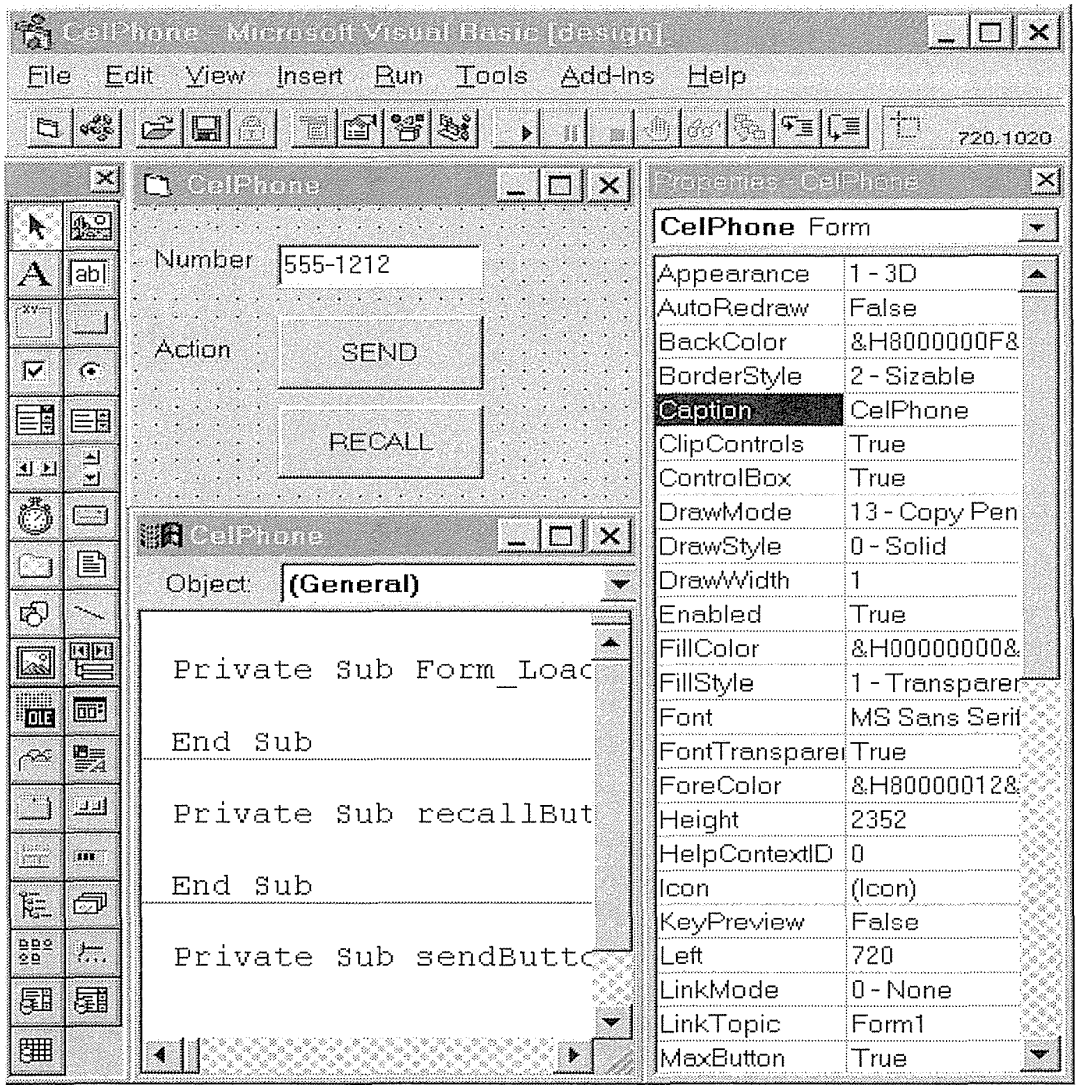


Figure 5.8

This Microsoft Visual Basic design shows a mock-up of a CelPhone interface with a text box for the phone number and two action buttons. The palette of tools on the left includes a Label, TextBox, Frame, CommandButton, CheckBox, RadioButton, ComboBox, ListBox, and scroll bars. The code window is in the bottom center and the properties window at the right allows users to set object properties. (Figures 5.8, 5.9 and 5.10 prepared by Stephan Greene, University of Maryland.) (Used with permission of Microsoft Corp., Redmond, WA.)

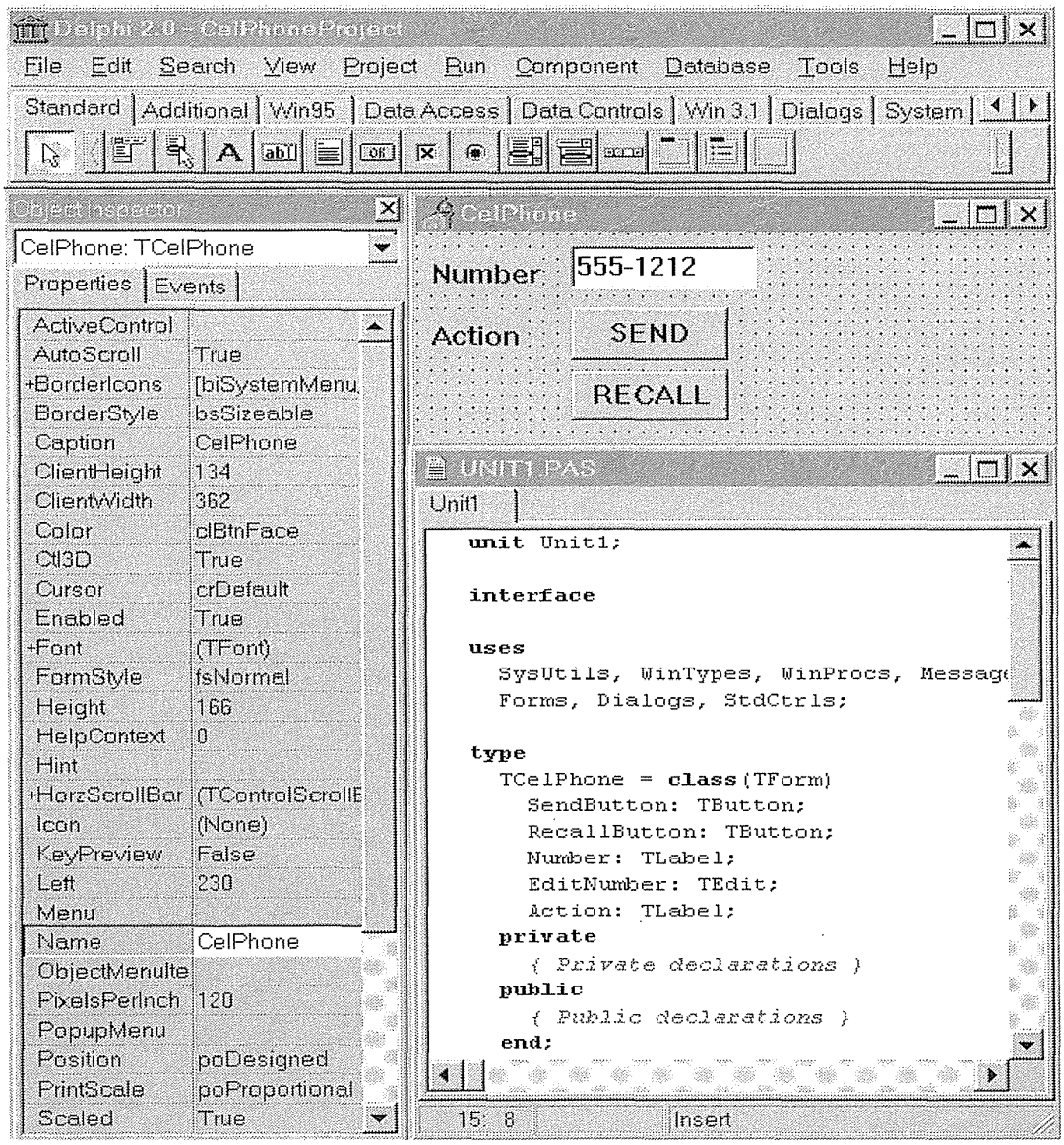


Figure 5.9

This Borland Delphi design shows the same mock-up of a CelPhone as in Fig. 5.8. The palette of tools, which is across the top, includes MainMenu, PopupMenu, Label, Edit, Memo Button, CheckBox, RadioButton, ListBox, ComboBox, ScrollBar, GroupBox, RadioGroup, and Panel. The Object Inspector window, which allows setting of properties, is at the left, and the code window is at the lower right. (Used with permission of Borland International, Inc., Scotts Valley, CA)

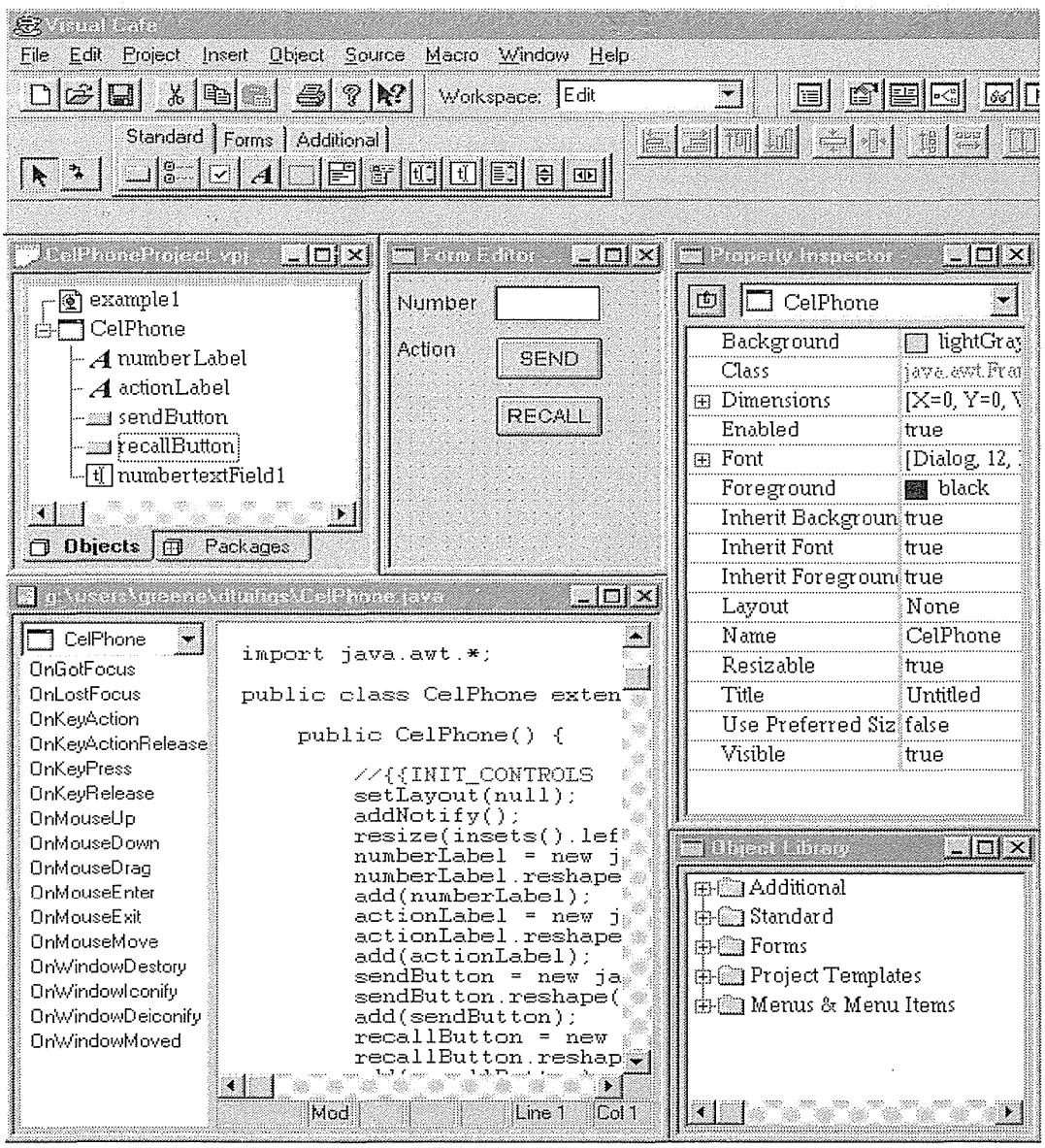


Figure 5.10

This Symantec Visual Cafe design shows the same mock-up of a CelPhone as in Fig. 5.8. The palette of tools, which is across the top, includes Button, RadioButton, CheckBox, Label, Panel, Choice, MenuBar, TextArea, TextField, List, Vertical Scrollbar, and Horizontal Scrollbar. The object hierarchy in the form is at the upper left, the code in the lower left, the properties window on the upper right, and the object library on the lower right. (Used with permission of Symantec Corp., Cupertino, CA.)

```
X/* Written by Dan Heller. Copyright 1991, O'Reilly & Associates.
X * This program is freely distributable without licensing fees and
X * is provided without guarantee or warrantee expressed or implied.
X * This program is -not- in the public domain.
======================================================================
X     /* main window contains a MenuBar and a Label displaying a pixmap
*/
X     main_w = XtVaCreateManagedWidget("main_window",
X         xMainWindowWidgetClass,    toplevel,
X         XmNscrollBarDisplayPolicy, XmAS_NEEDED,
X         XmNscrollingPolicy,        XmAUTOMATIC,
X         NULL);
X
X     /* Create a simple MenuBar that contains three menus */
X     file = XmStringCreateSimple("File");
X     edit = XmStringCreateSimple("Edit");
X     help = XmStringCreateSimple("Help");
X     menubar = XmVaCreateSimpleMenuBar(main_w, "menubar",
X         XmVaCASCADEBUTTON, file, 'F',
X         XmVaCASCADEBUTTON, edit, 'E'
X         XmVaCASCADEBUTTON, help, 'H',
X         NULL);
X    XmStringFree(file);
X    XmStringFree(edit);
X    /* don't free "help" compound string yet — reuse it for later */
X
X    /* Tell the menubar which button is the help menu */
X    if (widget - XtNameToWidget(menubar, "button_2"))
X        XtVaSetValues(menubar, XmNmenuHelpWidget, widget, NULL);
```

Figure 5.11

Programming of user interfaces in Motif.

```
#First make a menu button
menubutton .menu1 -text "Unix commands" -menu .menu1.m
-underline 0

#Now make the menu, and add the lines one at a time
menu .menu1.m
.menu1.m add command -label "List Files" -command {ls}
.menu1.m add command -label "Get date" -command {date}
.menu1.m add command -label "Start calendar" -command {xcalendar}

pack .menu1
```

A well-developed commercial alternative is Galaxy (Visix, Reston, VA), which offers cross-platform capability by emulating GUIs on Macintosh, Windows, Motif, and other platforms. The visual editor has rich functionality that

allows users to specify layouts with springs and struts to preserve the designer's intent even when screen sizes or widget sizes are changed (Hudson and Mohamed, 1990). Galaxy has rich object-oriented libraries that can be invoked from C or C++ programs, plus tools for managing network services and file directories. It requires software-engineering skills to use, but the visual editor enables prompt construction of prototypes.

Sun Microsystems has created the largest tremors on the web with its offerings of Java and Javascript. Java is a complete system-programming language that is specially designed for the World Wide Web. It is compiled on the server and is sent to clients as bytecodes that are interpreted by the browser on whatever platform the browser resides, thereby obtaining cross-platform capability. Java can be used to create complete applications that are distributed like any program, but one of its charms is its capacity to create "applets." These small program fragments can be downloaded from a web page and executed on the user's machine. This aspect enables programmers easily to make web pages dynamic and provide animations or error checking on data-entry forms. This extreme form of modularity allows software packages to be updated by way of the World Wide Web, and permits users to acquire only the components that they use.

Java is object oriented but eliminates some of the complexity of C++, such as operator overloading, multiple inheritance, pointers, and extensive automatic coercions. Automatic garbage collection and the absence of pointers eliminate common sources of bugs. Security and robustness goals were achieved by techniques such as strong typing, which requires explicit data declarations, and static binding, which means that references must be made during compilation. Software engineers have celebrated Java, because of its features and its familiar programming-language style, as indicated in this brief example from the online manual:

```
class Test {
   public static void main(String[] args) {
      for (int i = 0; i < args.length; i++)
          System.out.print(i == 0 ? args[i] : " " + args[i]);
          System.out.println();
      }
}
```

Javascript is a much simpler scripting language that is embedded in the Hypertext Markup Language (HTML) code that generates web pages. It achieves the goals of network distribution and cross-platform capability, since it is distributed within the HTML for a web page and is interpreted by the client's browser on the local machine—Macintosh, Windows, or Unix. It is relatively easy to learn, especially for someone who has learned HTML, and it supplies common features. This example shows a script to square the

value of a user-entered number:

```
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
<!-- to hide script contents from old browsers
  function square(i) {
    document.write("The call passed ", i ," to
      the function.",<BR>)
    return i * i
  }

  document.write("The function returned ",square(5),".")
// end hiding contents from old browsers  -->
</SCRIPT>
</HEAD>
<BODY>
<BR>
All done.
</BODY>
```

On loading the web page, it produces this output:

```
The call passed 5 to the function.
The function returned 25.
All done.
```

Although the original Java and Javascript did not contain visual editors, other developers will supply those tools. Security problems have arisen, but Java seems likely to provide adequate security to encourage development of commercial processes, such as funds transfer, credit-card charges, or personal data sharing. Execution speed of Java is a concern, because the bytecodes must be interpreted, but compilation techniques are promised to support rapid performance, and even hardware changes have been suggested.

The rapid pace of change on the Internet is stimulated by the easy sharing of code and the capacity to build quickly on top of the work of other programmers. The frenzy is sometimes alarming, but is usually irresistible. The importance of the World Wide Web has led developers of many tools—including Tcl/Tk, Galaxy, MacroMind Director, and Visual Basic—to enable their programs to run on the web.

5.4 Evaluation and Critiquing Tools

Software tools are natural environments in which to add procedures to evaluate or critique user interfaces. Simple metrics that report numbers of displays, widgets, or links between displays capture the size of a user-interface project. But the inclusion of more sophisticated evaluation procedures can

allow us to assess whether a menu tree is too deep or contains redundancies, whether widget labels have been used consistently, whether all buttons have proper transitions associated with them, and so on (Olsen and Halversen, 1988). Even straightforward tools such as spell checkers or concordances of terms would be a benefit.

A second set of tools is *run-time logging software*, which captures the users' patterns of activity. Simple reports—such as on the frequency of each error message, menu-item selection, dialog-box appearance, help invocation, form-field usage, or web-page access—are of great benefit to maintenance personnel and to revisers of the initial design. Experimental researchers can also capture performance data for alternative designs to guide their decision making. Software to analyze and summarize the performance data will be welcome.

An early example is Tullis' Display Analysis Program, which takes alphanumeric screen designs (no color, highlighting, separator lines, or graphics) and produces Tullis's display-complexity metrics plus some advice, such as this (Tullis, 1988):

```
Upper-case letters: 77% The percentage of upper-case letters
is high.
   Consider using more lower-case letters, since text printed
   in normal upper- and lower-case letters is read about 13%
   faster than text in all upper case. Reserve all upper-case
   for items that need to attract attention.

Maximum local density = 89.9% at row 9, column 8.
Average local density = 67.0%
   The area with the highest local density is identified
   ...you can reduce local density by distributing the
   characters as evenly as feasible over the entire
   screen.

Total layout complexity = 8.02 bits
Layout complexity is high.
   This means that the display items (labels and data) are
   not well aligned with each other...Horizontal complexity
   can be reduced by starting items in fewer different
   columns on the screen (that is, by aligning them verti-
   cally).
```

The movement toward GUIs with richer fonts and layout possibilities has reduced interest in Tullis's metrics, but better analyses of layouts seem possible (see Section 11.4). Evaluations based on formal user-task descriptions using NGOMSL (Byrne et al., 1994) or simpler task sequences and frequencies (Sears, 1993; 1995) are possible. Task-dependent metrics are likely to be more accurate, but the effort and uncertainty in collecting sequences and frequencies of tasks may discourage potential users.

Task-independent measurement and evaluation tools can be easily applied at low cost, early in the development process (Mahajan and Shneiderman, 1996). Simple measures such as the number of widgets per dialog box, widget density, nonwidget areas, aspect ratio, and balance of top to bottom or left to right are useful to gain some idea of the designer's style, but they have limited value in detecting anomalies. Reports on the top, bottom, left and right margins, and the list of distinct colors and typefaces often produced unreasonable variations in four systems developed using Visual Basic. Separate tools to perform spell checking and to produce interface concordances were helpful in revealing errors and inconsistencies. Software tools to check button size, position, color, and wording also revealed inconsistencies that were produced because multiple members of design teams failed to coordinate on a common style. An empirical study with 60 users demonstrated that increased variations in terminology—for example, switching from *search* to *browse* to *query*—slowed performance times by 10 to 25 percent.

Web-page and web-site analyzers also offer designers some guidance. Doctor HTML (http://imagiware.com/RxHTML/)provides link and spell checking; examines forms, tables, and images; and gives code evaluation with comments such as this:

```
Did not find the required open and close HEAD tag. You should
open and close the HEAD tag in order to get consistent per-
formance on all browsers. Found extra close STRONG tags in
this document. Please remove them.
```

5.5 Practitioner's Summary

There will always be a need to write some user interfaces with traditional programming tools, but the advantages of specialized user-interface software tools for designers and software engineers are large. They include an order-of-magnitude increase in productivity, shorter development schedules, support for expert reviews and usability testing, ease in making changes and ensuring consistency, better management control, and reduced training necessary for designers.

The profusion of current tools and the promises of improved tools requires that managers, designers, and programmers stay informed, and that they make fresh choices for each project. This educational process can be enlightening, since the benefits of improved and appropriate tools are enormous if the right tools are selected (Hix and Schulman, 1991) (Box 5.2).

From the tool maker's viewpoint, there are still great opportunities to create effective tools that handle more user-interface situations, that produce output for multiple software and hardware platforms, that are easier to learn, that are

Box 5.2

Factors in choosing among user-interface–building tools.

Widgets supported

- Windows and dialog boxes
- Pull-down or pop-up menus
- Buttons (rectangles, roundtangles, etc.)
- Radio buttons and switches
- Scroll bars (horizontal and vertical)
- Data-entry fields
- Field labels
- Boxes and separator lines
- Sliders, gauges, meters

Interface features

- Color, graphics, images, animation, video
- Varying display size (low to high resolution)
- Sounds, music, voice input–output
- Mouse, arrow keys, touchscreen, stylus

Software architecture

- Prototype only, prototype plus application-programming support, user-interface development environment
- Interface style (command language, menu, form fillin, or direct manipulation)
- Levels and strength of user-interface independence
- Programming language (specialized, standard (C, Pascal, etc.), visual)
- Evaluation and documentation tools
- Easy interface with database, graphics, networking, spreadsheets, etc.
- Logging during testing and use

Management issues

- Number of satisfied users of the tool
- Supplier reliability and stability
- Cost
- Documentation, training, and technical support
- Project-management support
- Integration with existing tools and processes

more powerful, and that provide more useful and accurate evaluation. Existing CASE tools could be expanded to include user-interface features.

5.6 Researcher's Agenda

The narrow focus of formal models of user interfaces and specification languages means that these models are beneficial for only small components. Scalable formal methods and automatic checking of user-interface features would be a major contribution. Innovative methods of specification involving graphical constraints or visual programming seem to be a natural match for creating GUIs. Improved software architectures are needed to ease the burden during revision and maintenance of user interfaces. Cooperative computing tools may provide powerful authoring tools that enable multiple designers to work together effectively on large projects. Other opportunities exist to create tools for designers of interfaces in novel environments using sound, animation, video, and virtual reality, and manipulating physical devices as in flexible manufacturing systems or home automation. Other challenges are to specify dynamic processes (gestural input), to handle continuous input (datastreams from a sensor), and to synchronize activities (to pop up a reminder box for 10 seconds, if a file has not been saved after 30 minutes of editing). As new interface styles emerge, there will always be a need to develop new tools to facilitate their construction. Metrics and evaluation tools are still open topics for user-interface and website developers. Specification by demonstration is an appealing notion (Myers, 1992), but practical application remains elusive.

World Wide Web Resources WWW

User interface tools are widely promoted on the web by companies and others. The World Wide Web is a great resource here because the technology changes so rapidly that books are immediately out of date. Online white papers, manuals, and tutorials are often effective and enable contact with developers. An imaginative idea is to have websites that will critique your website. Such online services are likely to expand in the coming years.

http://www.aw.com/DTUI

References

Carey, Tom, The gift of good design tools. In Hartson, H. R. and Hix, D. (Editors), *Advances in Human–Computer Interaction*, Volume II, Ablex, Norwood, NJ (1988), 175–213.

Byrne, Michael D., Wood, Scott D., Sukaviriya, Piyawadee "Noi," Foley, James D., and Kieras, David E., Automating interface evaluation, *Proc. CHI '94 Conference—Human Factors in Computing Systems*, ACM, New York (1994), 232–237.

Carr, David, Specification of interface interaction objects, *Proc. CHI '94 Conference—Human Factors in Computing Systems*, ACM, New York (1994), 372–378.

Chase, J. D., Schulman, Robert S., Hartson, H. Rex, and Hix, Deborah, Development and evaluation of a taxonomical model of behavioral representation techniques, *Proc. CHI '94 Conference—Human Factors in Computing Systems*, ACM, New York (1994), 159–165.

Green, Thomas R. G. and Petre, Marian, Usability analysis of visual programming environments: A "cognitive dimensions" framework, *Journal of Visual Languages and Computing*, 7, (1996), 131–174.

Harel, David, On visual formalisms, *Communications of the ACM*, 31, 5 (May 1988), 514–530.

Hartson, H. Rex, Siochi, Antonio C., and Hix, Deborah, The UAN: User-oriented representation for direct manipulation interface designs, *ACM Transactions on Information Systems*, 8, 3 (July 1990), 181–203.

Hix, Deborah and Hartson, H. Rex, *Developing User Interfaces: Ensuring Usability Through Product and Process*, John Wiley and Sons, New York (1993).

Hix, Deborah and Schulman, Robert S., Human–computer interface development tools: A methodology for their evaluation, *Communications of the ACM*, 34, 3 (March 1991), 74–87.

Hudson, Scott E. and Mohamed, Shamim P., Interactive specification of flexible user interface displays, *ACM Transactions on Information Systems*, 8, 3 (July 1990), 269–288.

Jacob, Robert J. K., An executable specification technique for describing human–computer interaction. In Hartson, H. Rex (Editor), *Advances in Human–Computer Interaction*, Volume I, Ablex, Norwood, NJ (1985), 211–242.

Mahajan, Rohit and Shneiderman, Ben, Visual and textual consistency checking tools for graphical user interfaces, Dept. of Computer Science Tech Report CS-TR-3639, University of Maryland, College Park, MD (1996).

Myers, Brad A., Demonstrational interfaces: A step beyond direct manipulation, *IEEE Computer*, 25, 8 (August 1992), 61–73.

Myers, Brad A., User interface software tools, *ACM Transactions on Computer–Human Interaction*, 2, 1 (March 1995), 64–103.

Olsen, Jr., Dan R., *User Interface Management Systems: Models and Algorithms*, Morgan Kaufmann Publishers, San Mateo, CA (1991).

Olsen, Jr., Dan R. and Halversen, Bradley W., Interface usage measurement in a User Interface Management System, *Proc. ACM SIGGRAPH Symposium on User Interface Software and Technology*, ACM Press, New York (1988), 102–108.

Ousterhout, John, *Tcl and the Tk Toolkit*, Addison-Wesley, Reading, MA (1994).

Reisner, Phyllis, Formal grammar and design of an interactive system, *IEEE Transactions on Software Engineering*, SE-5, (1981), 229–240.

Shneiderman, Ben, Multi-party grammars and related features for defining interactive systems, *IEEE Systems, Man, and Cybernetics*, SMC-12, 2 (March–April 1982), 148–154.

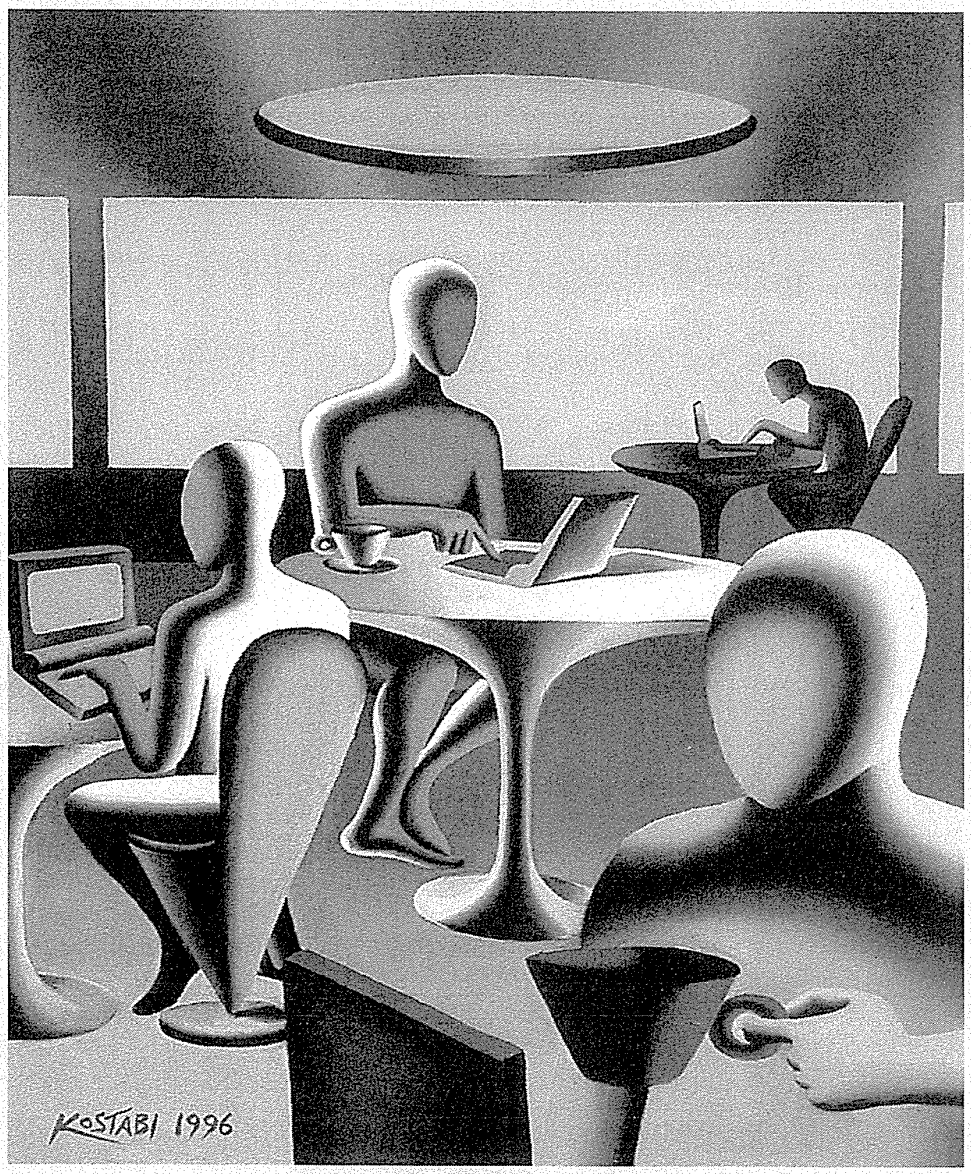
Sears, Andrew, Layout appropriateness: Guiding user interface design with simple task descriptions, *IEEE Transactions on Software Engineering*, 19, 7 (1993), 707–719.

Sears, Andrew, AIDE: A step towards metrics-based interface development tools, *Proc. UIST '95 User Interface Software and Technology*, ACM, New York (1995), 101–110.

Tullis, Thomas, A system for evaluating screen formats: research and application. In Hartson, H. Rex and Hix, D. (Editors), *Advances in Human–Computer Interaction*, Volume II, Ablex, Norwood, NJ (1988), 214–286.

Wasserman, Anthony I., and Shewmake, David T., The role of prototypes in the User Software Engineering (USE) methodology. In Hartson, Rex (Editor), *Advances in Human–Computer Interaction*, Volume I, Ablex, Norwood, NJ (1985), 191–210.

Wellner, Pierre D., Statemaster: A UIMS based on statecharts for prototyping and target implementation, *Proc. CHI '89 Conference—Human Factors in Computing Systems*, ACM, New York (1989), 177–182.



Mark Kostabi, *Computer Cafe (Uploading the Future)*, 1996

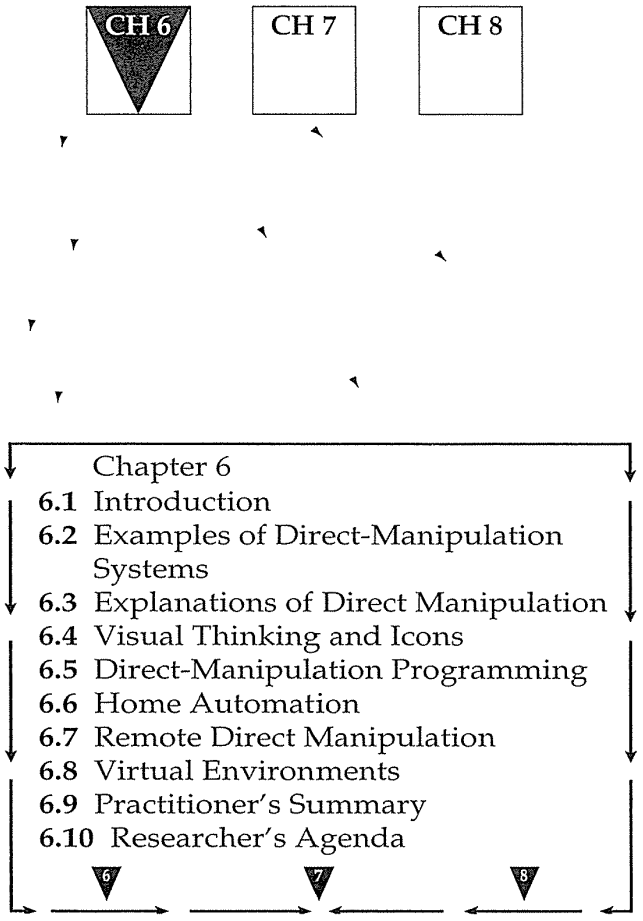
CHAPTER

6

Direct Manipulation and Virtual Environments

Leibniz sought to make the form of a symbol reflect its content. "In signs," he wrote, "one sees an advantage for discovery that is greatest when they express the exact nature of a thing briefly and, as it were, picture it; then, indeed, the labor of thought is wonderfully diminished."

Frederick Kreiling, "Leibniz,"
***Scientific American*, May 1968**



6.1 Introduction

Certain interactive systems generate a glowing enthusiasm among users that is in marked contrast with the more common reaction of grudging acceptance or outright hostility. The enthusiastic users report the following positive feelings:

- Mastery of the interface
- Competence in performing tasks
- Ease in learning the system originally and in assimilating advanced features
- Confidence in the capacity to retain mastery over time

- Enjoyment in using the system
- Eagerness to show off the system to novices
- Desire to explore more powerful aspects of the system

These feelings convey an image of the truly pleased user. The central ideas in the systems that inspire such delight are visibility of the objects and actions of interest; rapid, reversible, incremental actions; and replacement of complex command-language syntax by direct manipulation of the object of interest (Shneiderman, 1983). The objects–actions interface (OAI) model provides a sound foundation for understanding direct manipulation since it steers designers to represent the task domain objects and actions while minimizing the interface concepts and the syntax-memorization load. Direct-manipulation thinking has spawned the new strategies of information visualization (see Chapter 15) that present thousands of objects on the screen with dynamic user controls.

6.2 Examples of Direct-Manipulation Systems

No single system has every admirable attribute or design feature—such a system might not be possible. Each of the following examples, however, has sufficient numbers of them to win the enthusiastic support of many users.

My favorite example of using direct manipulation is driving an automobile. The scene is directly visible through the front window, and performance of actions such as braking or steering has become common knowledge in our culture. To turn left, the driver simply rotates the steering wheel to the left. The response is immediate and the scene changes, providing feedback to refine the turn. Imagine trying to turn by issuing a command LEFT 30 DEGREES and then another command to see the new scene; but that is the level of operation of many office-automation tools of today! Another well-established example is air-traffic control, in which users see a representation of the airspace with brief data blocks attached to each plane. Controllers move a trackball to point at specific planes and to perform actions.

6.2.1 Command-line versus display editors versus word processors

It may be hard for new users of word processors to believe, but in the early 1980s, text editing was done with line-oriented command languages. Users might see only one line at a time and typed commands were needed to move the view window or to make any changes. The users of novel *full-page display editors* were great advocates of their systems. A typical comment was, "Once you've used a display editor, you will never want to go back to a line editor—

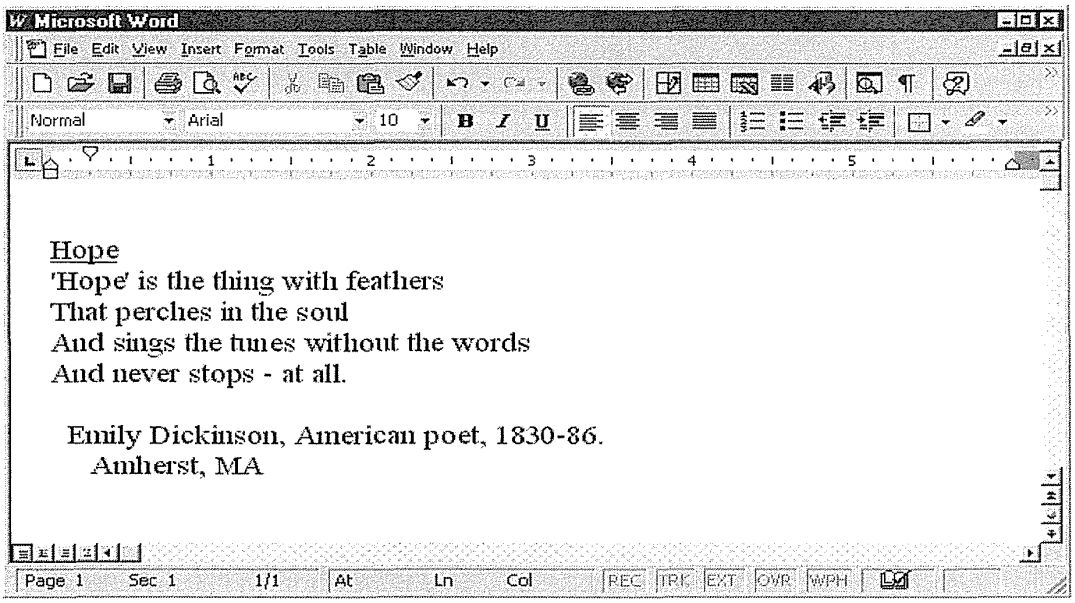


Figure 6.1

An example of a WYSIWYG (What You See Is What You Get) editor: Microsoft Word for Office 97. (Used with permission of Microsoft Corp., Redmond, WA.)

you'll be spoiled." Similar comments came from users of early personal-computer word processors, such as WORDSTAR, or display editors such as emacs or vi (for visual editor) on the Unix system. A beaming advocate called emacs "the one true editor." In these systems users viewed up to a full screen of text and could edit by using backspace or insert directly by typing.

Researchers found that performance was improved and training times were reduced with display editors so there was evidence to support the enthusiasm of display-editor devotees. Furthermore, office-automation evaluations consistently favored full-page display editors for secretarial and executive use. There are some advantages to command-language approaches, such as that history keeping is easier, more flexible markup languages are available (for example, SGML), macros tend to be more powerful, and some tasks are simpler to express (for example, change all italics to bold). Strategies for accommodating these needs are finding their way into modern direct-manipulation word processors.

By the early 1990s, *what you see is what you get (WYSIWYG)* word processors had become standard. Microsoft Word has become dominant on the Macintosh and IBM PC compatibles, with Lotus Word Pro and Corel's WordPerfect taking second place (Fig. 6.1). The advantages of WYSIWYG word processors include the following:

- *Display a full page of text* Showing 20 to 60 lines of text simultaneously gives the reader a clearer sense of context for each sentence, while permitting simpler reading and scanning of the document. By contrast, working with the one-line-at-a-time view offered by line editors is like seeing the world through a narrow cardboard tube. Some large displays can support two full pages of text, set side by side.
- *Display the document in the form that it will appear when the final printing is done* Eliminating the clutter of formatting commands also simplifies reading and scanning of the document. Tables, lists, page breaks, skipped lines, section headings, centered text, and figures can be viewed in their final form. The annoyance and delay of debugging the format commands are almost eliminated because the errors are usually apparent immediately.
- *Show cursor action to the user* Seeing an arrow, underscore, or blinking box on the screen gives the operator a clear sense of where to focus attention and to apply action.
- *Control cursor motion through physically obvious and intuitively natural means* Arrow keys or cursor-motion devices—such as a mouse, joystick, or graphic tablet—provide natural physical mechanisms for moving the cursor. This setup is in marked contrast to commands, such as UP 6, that require an operator to convert the physical action into a correct syntactic form that may be difficult to learn and hard to recall, and thus may be a source of frustrating errors.
- *Use labeled icons for actions* Most word processors have labeled icons in a toolbar for frequent actions. These buttons act as a permanent menu-selection display to remind users of the features and to provide rapid selection.
- *Display the results of an action immediately* When the user presses a button to move the cursor or center text, the results are shown immediately on the screen. Deletions are apparent immediately: the character, word, or line is erased, and the remaining text is rearranged. Similarly, insertions or text movements are shown after each keystroke or function-key press. In contrast, with line editors, users must issue print or display commands to see the results of changes.
- *Provide rapid response and display* Most display editors operate at high speed; a full page of text appears in a fraction of a second. This high display rate coupled with short response time produces a satisfying sense of power and speed. Cursors can be moved quickly, large amounts of text can be scanned rapidly, and the results of actions can be shown almost instantaneously. Rapid response also reduces the need for additional commands and thereby simplifies design and learning. Line editors with slow display rates and long response times bog down the user. Speeding up line editors would add to their attrac-

tiveness, but they would still lack such features as direct overtyping, deletion, and insertion.

- *Offer easily reversible actions* When users enter text, they repair an incorrect keystroke by merely backspacing and overstriking. They can make simple changes by moving the cursor to the problem area and overstriking, inserting, or deleting characters, words, or lines. A useful design strategy is to include natural inverse operations for each operation (for example, to increase or decrease type sizes). An alternative offered by many display editors is a simple UNDO command to return the text to the state that it was in before the previous action or action sequence. The easy reversibility reduces user anxiety about making a mistake or destroying the file.

So many of these issues have been studied empirically that someone joked that the word processor is the white rat for researchers in human–computer interaction. Switching metaphors, for commercial developers, we might say the word processor is the root for many technological sprouts:

- *Integration* of graphics, spreadsheets, animations, photographs, and so on is done in the body of a document. Advanced designs, such as the OpenDoc, even permit "hot links" so that, if the graphic or spreadsheet is changed, the copy in the document also will be changed.
- *Desktop-publishing software* produces sophisticated printed formats with multiple columns and allows output to high-resolution printers. Multiple fonts, grayscales, and color permit preparation of high-quality documents, newsletters, reports, newspapers, or books. Examples include Adobe PageMaker and QuarkXPress.
- *Slide-presentation software* produces color text and graphic layouts for use as overhead transparencies, 35-millimeter slides, or directly from the computer with a large-screen projector to allow animations.
- *Hypermedia environments* with selectable buttons or embedded menu items allow users to jump from one article to another. Links among documents, bookmarks, annotations, and tours can be added by readers.
- *Improved macro facilities* enable users to construct, save, and edit sequences of frequently used actions. A related feature is a style sheet that allows users to specify and save a set of options for spacing, fonts, margins, and so on. Another feature is the saving of templates that allows users to take the formatting work of colleagues as a starting point for their own documents. Most word processors come with dozens of standard templates for business letters, newsletters, or brochures.
- *Spell checkers and thesauri* are standard on most full-featured word processors. Spell checking can also be set to function while the user is

typing, or to make automatic changes for common mistakes, such as changing "teh" to "the."

- *Grammar checkers* offer users comments about potential problems in writing style, such as use of passive voice, excessive use of certain words, or lack of parallel construction. Some writers—both novices and professionals—appreciate the comments and know that they can decide whether to apply the suggestions. Critics point out, however, that the advice is often inappropriate and therefore wastes time.
- *Document assemblers* allow users to compose complex documents, such as contracts or wills, from standard paragraphs using appropriate language for males or females; citizens or foreigners; high, medium, or low income earners; renters or home owners, and so on.

6.2.2 The VisiCalc spreadsheet and its descendants

The first electronic spreadsheet, VisiCalc, was the product of a Harvard Business School student, Dan Bricklin, who became frustrated when trying to carry out repetitious calculations for a graduate course in business. He and a friend, Bob Frankston, built an "instantly calculating electronic worksheet" (as the user manual described it) that permitted computation and immediate display of results across 254 rows and 63 columns.

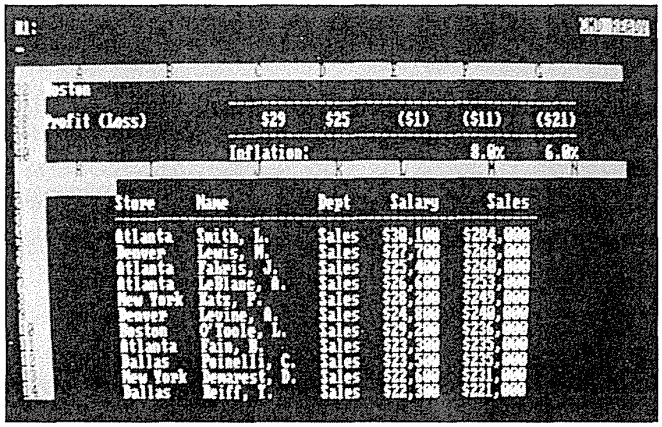
The *spreadsheet* can be programmed so that column 4 displays the sum of columns 1 through 3; then, every time a value in the first three columns changes, the fourth column changes as well. Complex dependencies among manufacturing costs, distribution costs, sales revenue, commissions, and profits can be stored for several sales districts and for various months, so that the effects of changes on profits can be seen immediately.

This simulation of an accountant's spreadsheet makes it easy for novices to comprehend the objects and permissible actions. Spreadsheet users can try out alternate plans and see the effects on sales or profit. The distributor of VisiCalc explained the system's appeal as "it jumps," referring to the user's delight in watching the propagation of changes across the screen.

Competitors to VisiCalc emerged quickly; they made attractive improvements to the user interface and expanded the tasks that were supported. LOTUS 1-2-3 dominated the market in the 1980s (Fig. 6.2), but the current leader is Microsoft's Excel (Fig. 6.3), which has a large number of features and specialized additions. Excel and other modern spreadsheets offer integration with graphics, three-dimensional representations, multiple windows, and database features. The features are invoked easily with command menus or toolbars, and can be used within powerful macro facilities.

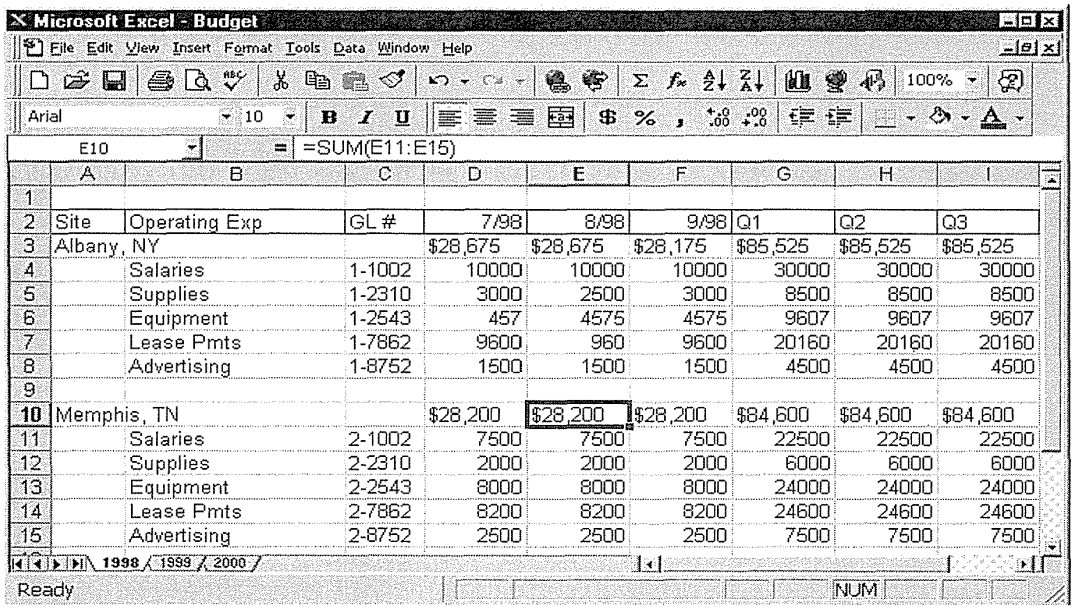
Figure 6.2

Early version of Lotus 1-2-3, the spreadsheet program that was dominant through the 1980s. (Printed with permission of Lotus Development Corporation, Cambridge, MA.)



6.2.3 Spatial data management

In geographic applications, it seems natural to give a spatial representation in the form of a map that provides a familiar model of reality. The developers of the prototype Spatial Data Management System (Herot, 1980; 1984) attribute the basic idea to Nicholas Negroponte of MIT. In one early scenario, the user was seated before a color-graphics display of the world and could



	A	B	C	D	E	F	G	H	I
1									
2	Site	Operating Exp	GL #	7/98	8/98	9/98	Q1	Q2	Q3
3	Albany, NY			$28,675	$28,675	$28,175	$85,525	$85,525	$85,525
4		Salaries	1-1002	10000	10000	10000	30000	30000	30000
5		Supplies	1-2310	3000	2500	3000	8500	8500	8500
6		Equipment	1-2543	457	4575	4575	9607	9607	9607
7		Lease Pmts	1-7862	9600	960	9600	20160	20160	20160
8		Advertising	1-8752	1500	1500	1500	4500	4500	4500
9									
10	Memphis, TN			$28,200	$28,200	$28,200	$84,600	$84,600	$84,600
11		Salaries	2-1002	7500	7500	7500	22500	22500	22500
12		Supplies	2-2310	2000	2000	2000	6000	6000	6000
13		Equipment	2-2543	8000	8000	8000	24000	24000	24000
14		Lease Pmts	2-7862	8200	8200	8200	24600	24600	24600
15		Advertising	2-8752	2500	2500	2500	7500	7500	7500

Figure 6.3

Microsoft Excel spreadsheet for Office 97. (Used with permission of Microsoft Corp., Redmond, WA.)

zoom in on the Pacific Ocean to see markers for convoys of military ships (Fig. 6.4). By moving a joystick, the user caused the screen to be filled with silhouettes of individual ships; zooming displayed detailed data, such as, ultimately, a full-color picture of the captain.

In another scenario, icons representing such different aspects of a corporation as personnel, an organizational chart, travel information, production data, and schedules were shown on a screen. By moving the joystick and zooming in on objects of interest, the user could travel through complex *information spaces (I-spaces)* to locate the item of interest. A building floorplan showing departments might be displayed; when a department was chosen, individual offices became visible. As the cursor was moved into a room, details of the occupant appeared on the screen. If users chose the wrong room, they merely backed out and tried another. The lost effort was minimal, and there was no stigma attached to error. The recent Xerox PARC Information Visualizer is an ensemble of tools that permit three-dimensional animated explorations of buildings, cone-shaped file directories, organization charts, a perspective wall that puts featured items up front and centered, and several two- and three-dimensional information layouts (Robertson et al., 1993).

ArcView (ESRI, Inc.) is a widely used geographic-information system that offers rich, layered databases of map-related information (Fig. 6.5). Users can zoom in on areas of interest, select the kinds of information they wish to view (roads, population density, topography, rainfall, political boundaries, and much more), and do limited searches. Much simpler but widely popular highway maps are available for the entire United States on a single CD-ROM. Map servers on the World Wide Web are increasingly popular for taking tours of cities, checking weather reports, or buying a home.

The success of a spatial data-management system depends on the skill of the designers in choosing icons, graphical representations, and data layouts that are natural and comprehensible to the user. The joy of zooming in and out, or of gliding over data with a joystick, entices even anxious users, who quickly demand additional power and data.

6.2.4 Video games

For many people, the most exciting, well-engineered, and commercially successful application of the concepts that we have been discussing lies in the world of video games (Provenzo, 1991). The early but simple and popular game PONG required the user to rotate a knob that moved a white rectangle on the screen. A white spot acted as a ping-pong ball that ricocheted off the wall and had to be hit back by the movable white rectangle. Users developed speed and accuracy in placing the "paddle" to keep the increasingly speedy ball from getting past, while the computer speaker emitted a ponging sound

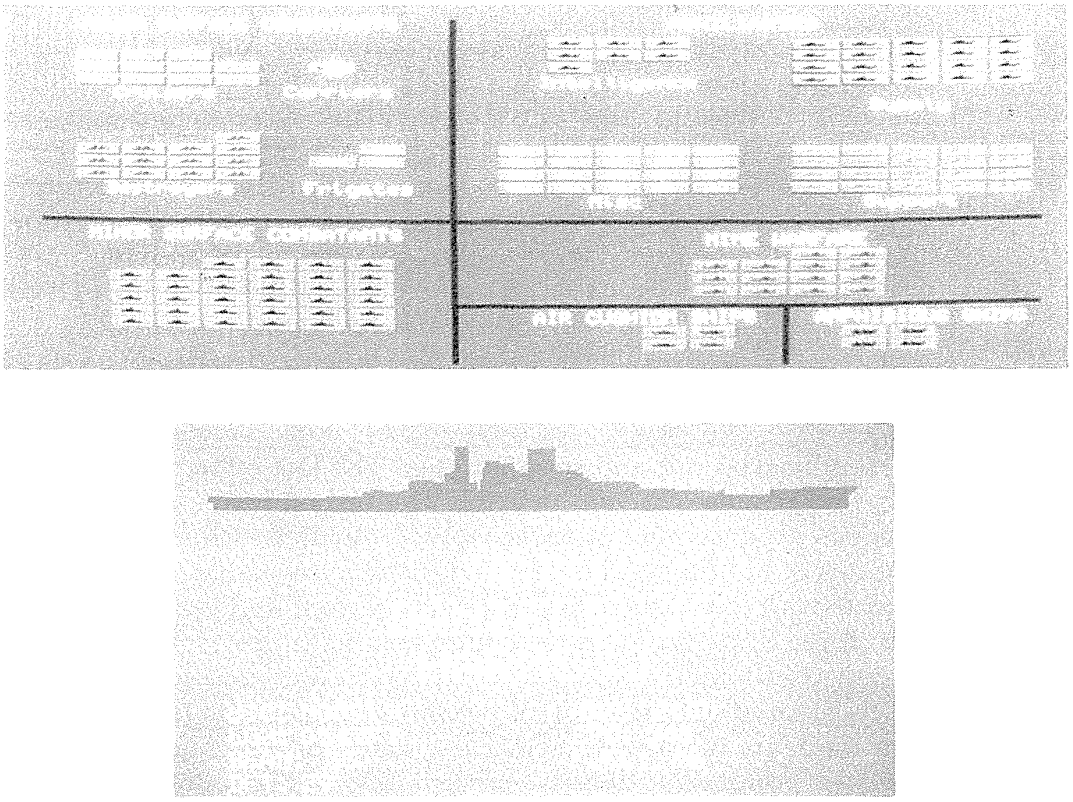


Figure 6.4

The Spatial Data Management System. Three displays to show multiple levels of detail or related information. The user moves a joystick to traverse information spaces or to zoom in on a map to see more details about ship convoys. (Courtesy of the Computer Corporation of America, Cambridge, MA.)

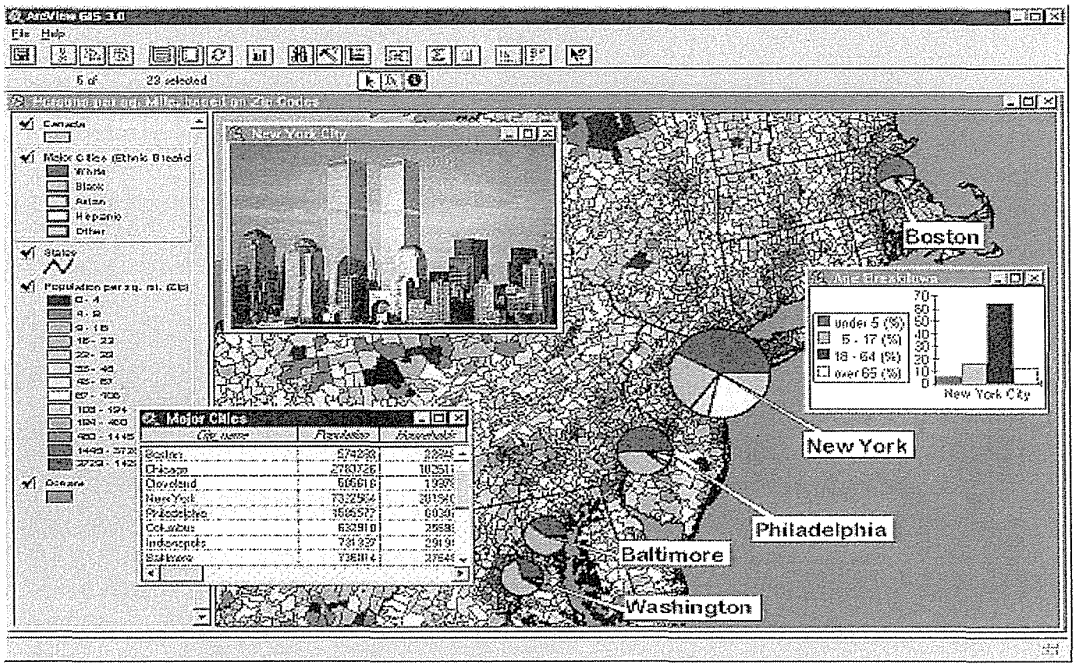


Figure 6.5

ArcView® geographic information system (GIS), which provides comprehensive mapping functions and management of related data. This map of the northeast United States shows color coding by population density for each zip code, ethnic makeup of large cities, and a photo of New York City. (Graphic image supplied courtesy of Environmental Systems Research Institute, Inc., Redlands, CA. Copyright 1996.)

when the ball bounced. Watching someone else play for 30 seconds is all the training that a person needs to become a competent novice, but many hours of practice are required to become a skilled expert.

Later games, such as Missile Command, Donkey Kong, Pac Man, Tempest, TRON, Centipede, or Space Invaders, were much more sophisticated in their rules, color graphics, and sound effects. Recent games include multiperson competitions in tennis or karate, three-dimensional graphics, still higher resolution, and stereo sound. The designers of these games provide stimulating entertainment, a challenge for novices and experts, and many intriguing lessons in the human factors of interface design—somehow, they have found a way to get people to put quarters in the sides of computers. Forty-million Nintendo game players reside across 70 percent of those American households that include 8 to 12 year olds. Brisk sales of the Mario series testify to the games' strong attraction, in marked contrast to the anxiety about and resistance to office-automation equipment that many users have

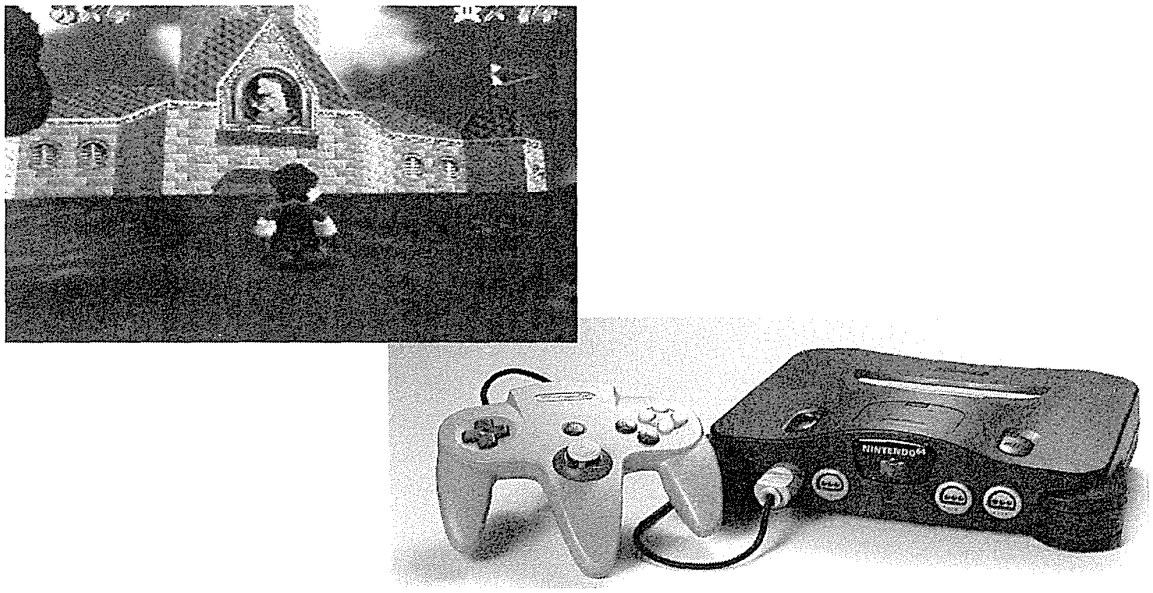


Figure 6.6

Home videogames are a huge success and employ advanced graphics hardware for rapid movement in rich three-dimensional worlds. The Nintendo 64 player can be used with a variety of games including the popular Super Mario® series (© 1997 Nintendo. Images courtesy of Nintendo of America Inc.)

shown (Fig. 6.6). The SEGA game player, Nintendo 64, and Sony Playstation have brought powerful three-dimensional graphics hardware to the home and have created a remarkable international market. Small hand-held game devices, such as the Game Boy®, provide portable fun for kids on the street or executives in their 30,000-foot-high offices.

These games provide a field of action that is visual and compelling. The commands are physical actions—such as button presses, joystick motions, or knob rotations—whose results are shown immediately on the screen. There is no syntax to remember, and therefore there are no syntax-error messages. If users move their spaceships too far left, then they merely use the natural inverse operation of moving back to the right. Error messages are unnecessary, because the results of actions are obvious and can be reversed easily. These principles can be applied to office automation, personal computing, or other interactive environments.

Most games continuously display a numeric score so that users can measure their progress and compete with their previous performance, with friends, or with the highest scorers. Typically, the 10 highest scorers get to store their initials in the game for public display. This strategy provides one form of positive reinforcement that encourages mastery. Malone's (1981), Provenzo's (1991), and our studies with elementary-school children

have shown that continuous display of scores is extremely valuable. Machine-generated feedback—such as "Very good" or "You're doing great!"—is not as effective, since the same score carries different meanings for different people. Most users prefer to make their own subjective judgments and perceive the machine-generated messages as an annoyance and a deception.

Many educational games use direct manipulation effectively. Elementary- or high-school students can learn about urban planning by using SimCity and its variants, which show urban environments visually and let students build roads, airports, housing, and so on by direct-manipulation actions.

The esthetically appealing MYST and its successor Riven (Broderbund) have drawn widespread approval even in some literary circles, while the more violent but wildly successful DOOM series has provoked controversy over its psychological effects on teens. Studying game design is fun, but there are limits to the applicability of the lessons. Game players seek entertainment and focus on the challenge of mastery, whereas applications users focus on their task and may resent too many playful distractions. The random events that occur in most games are meant to challenge the user; in nongame designs, however, predictable system behavior is preferred. Game players are engaged in competition with the system or with other players, whereas applications-systems users prefer a strong internal locus of control, which gives them the sense of being in charge.

6.2.5 Computer-aided design

Many *computer-aided design (CAD)* systems for automobiles, electronic circuitry, architecture, aircraft (see Fig. 1.3), or slide layout (Fig. 6.7) use principles of direct manipulation. The operator may see a circuit schematic on the screen and, with mouse clicks, be able to move resistors or capacitors into or out of the proposed circuit. When the design is complete, the computer can provide information about current, voltage drops, and fabrication costs, and warnings about inconsistencies or manufacturing problems. Similarly, newspaper-layout artists or automobile-body designers can easily try multiple designs in minutes, and can record promising approaches until they find even better ones.

The pleasures in using these systems stem from the capacity to manipulate the object of interest directly and to generate multiple alternatives rapidly. Some systems have complex command languages; most have moved to using cursor action and graphics-oriented commands.

Related applications are *computer-aided manufacturing (CAM)* and process control. Honeywell's process-control system provides the manager of an oil refinery, paper mill, or power-utility plant with a colored schematic view of

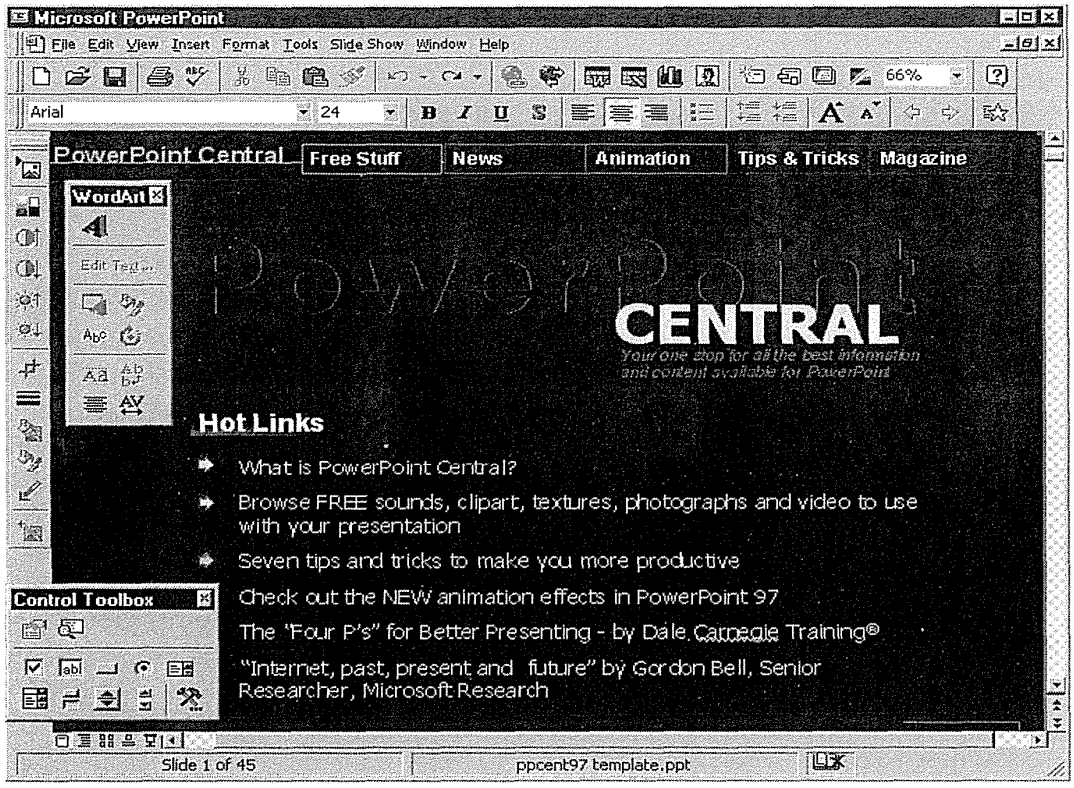


Figure 6.7

Presentation graphics or slide programs, such as Microsoft's PowerPoint for Office 97, have multiple toolbars and palettes that support a direct-manipulation style of selecting objects, moving them, and resizing them. (Used with permission of Microsoft Corp., Redmond, WA.)

the plant. The schematic may be displayed on eight displays or on a large wall-sized map, with red lines indicating a sensor value that is out of normal range. With a single click, the operator can get a more detailed view of the troubling component; with a second click, the operator can examine individual sensors or can reset valves and circuits.

A basic strategy for this design is to eliminate the need for complex commands that the operator would need to recall during a once-a-year emergency. The visual overview provided by the schematic facilitates problem solving by analogy, since the linkage between the screen representations and the plant's temperatures or pressures is so close.

6.2.6 Office automation

Designers of advanced *office-automation systems* used direct-manipulation principles. The pioneering Xerox Star (Smith et al., 1982) offered sophisticated text-formatting options, graphics, multiple fonts, and a high-resolution, cursor-based user interface (Fig. 6.8). Users could move (but not drag) a document icon to a printer icon to generate a hardcopy printout. The Apple Lisa system elegantly applied many of the principles of direct manipulation; although it was not a commercial success, it laid the groundwork for the successful Macintosh. The Macintosh designers drew from the Star and Lisa experiences, but made many simplifying decisions while preserving adequate power for users (Fig. 6.9). The hardware and software designs supported rapid and continuous graphical interaction for pull-down menus, window manipulation, editing of graphics and text, and dragging of icons. Variations on the Macintosh appeared soon afterward for other popular personal computers, and by now Windows 95 dominates the office-automation market (Color Plate 1). The Windows 95 design is still a close relative of the Macintosh design, and both are candidates for substantial improvements in window management (Chapter 13), with simplifications for novices and increased power for sophisticated users.

Studies of users of direct-manipulation interfaces have confirmed the advantages for at least some users and tasks. In a study of 30 novices, MS-DOS commands for creating, copying, renaming, and erasing files were contrasted with Macintosh direct-manipulation actions. After user training and practice, average task times were 5.8 minutes versus 4.8 minutes, and average errors were 2.0 versus 0.8 (Margono and Shneiderman, 1987). Subjective preference also favored the direct-manipulation interface. In a study of a command-line versus a direct-manipulation database interface, 55 "computer naive but keyboard literate" users made more than twice as many errors with the command-line format. No significant differences in time were found (Morgan et al., 1991). These users preferred the direct-manipulation interface overall, and rated it as more stimulating, easier, and more adequately powerful. Both reports caution about generalizing their results to more experienced users. A study with novices and experienced users was cosponsored by Microsoft and Zenith Data Systems (Temple, Barker, and Sloane, Inc., 1990). Although details about subjects, interfaces, and tasks were not reported, the results showed improved productivity and reduced fatigue for experienced users with a GUI, as compared with a character-based user interface. The benefits of direct manipulation were confirmed in other studies (Benbasat and Todd, 1993); one such study also demonstrated that the advantage was greater for experienced than for novice users (Ulich et al., 1991).

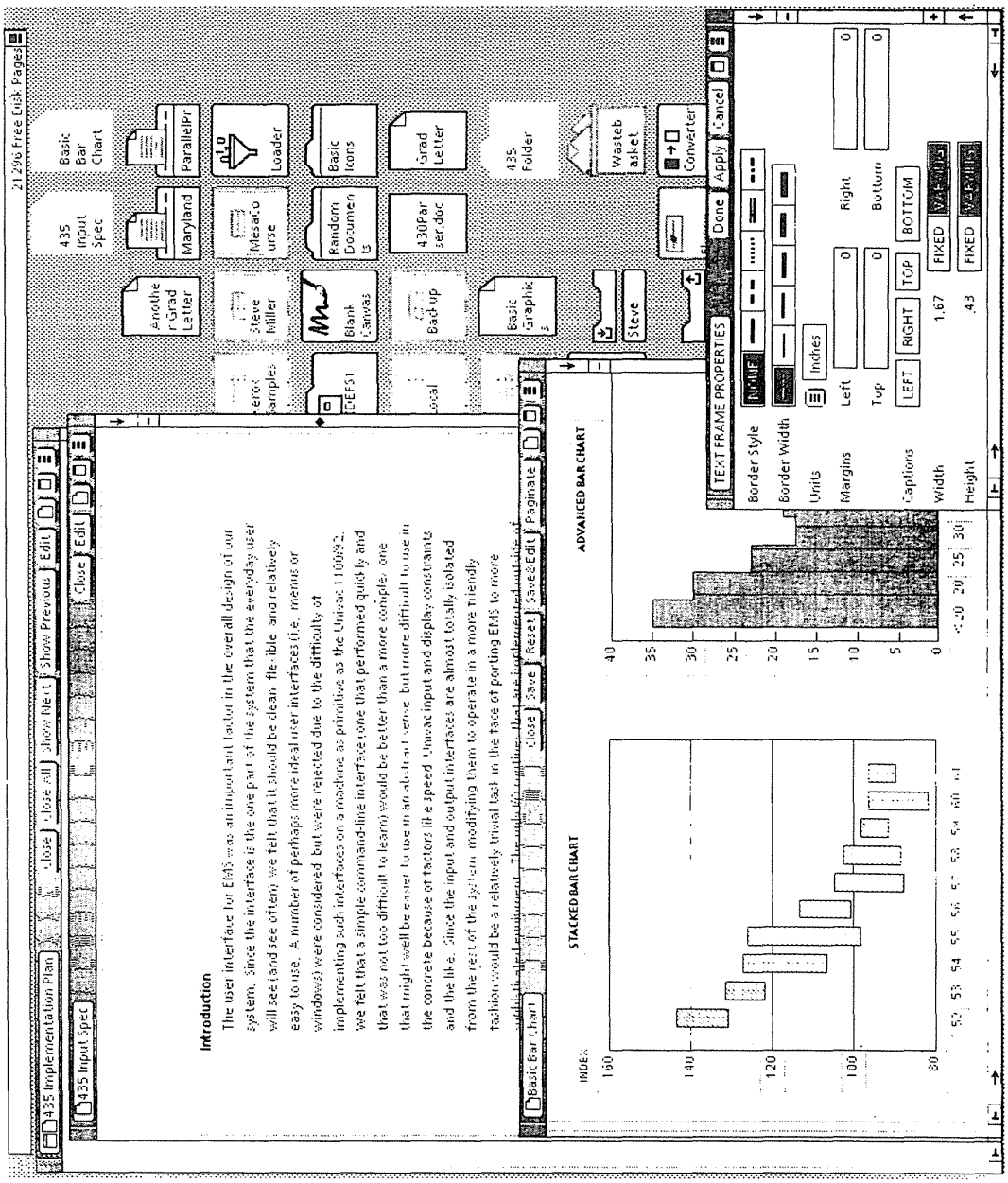


Figure 6.8

The Xerox Star 8010 with the ViewPoint system enables users to create documents with multiple fonts and graphics. This session shows the Text Frame Properties sheet over sample bar charts, with a document in the background and many desktop icons available for selection. (Prepared by Steve Miller, University of Maryland.)

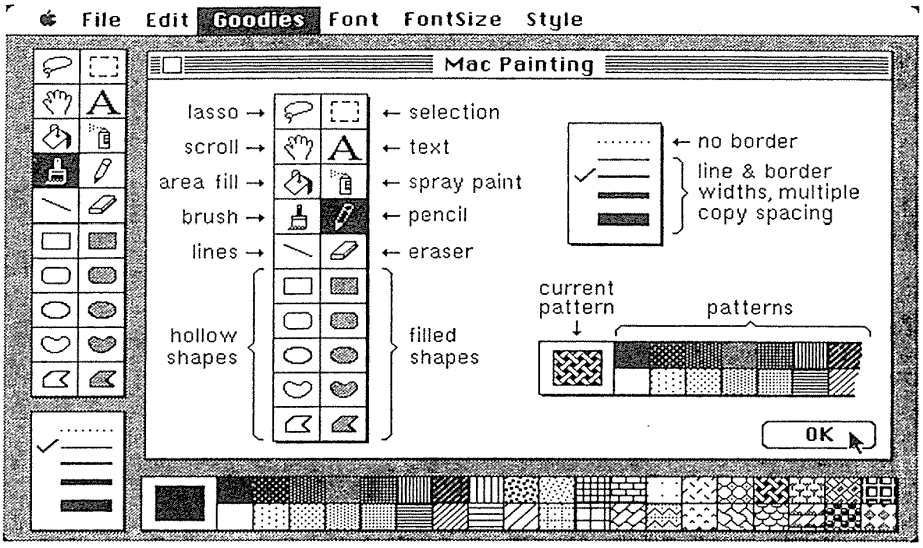


Figure 6.9

The original Apple Macintosh MacPaint. This program offers a command menu on the top, a menu of action icons on the left, a choice of line thicknesses on the lower left, and a palette of texture on the bottom. All actions can be accomplished with only the mouse. (Photo courtesy of Apple Computer, Inc., Cupertino, CA)

6.2.7 Further examples of direct manipulation

The trick in creating a direct-manipulation system is to come up with an appropriate representation or model of reality. Some designers may find it difficult to think about information problems in a visual form; with practice, however, they may find it more natural. With many applications, the jump to visual language may be difficult; later, however, users and designers can hardly imagine why anyone would want to use a complex syntactic notation to describe an essentially visual process. In fact, it is hard to think of new command languages developed after 1990. It is hard to conceive of learning the commands for the vast number of features in modern word processors, drawing programs, or spreadsheets, but the visual cues, icons, menus, and dialog boxes make it possible for even intermittent users to succeed.

Several designers applied direct manipulation using the metaphor of a stack of cards to portray a set of addresses, telephone numbers, events, and so on. Clicking on a card brings it to the front, and the stack of cards moves to preserve alphabetic ordering. This simple card-deck metaphor, combined with other notions (Heckel, 1991) led to Bill Atkinson's innovative development of HyperCard stacks in 1987 (see Section 15.4). Billed as a way to "create your own applications for gathering, organizing, presenting, searching,

and customizing information," HyperCard quickly spawned variants, such as SuperCard and ToolBook. Each has a scripting language to enable users to create appealing graphics applications.

Direct-manipulation checkbook-maintenance and checkbook-searching interfaces, such as Quicken (Intuit, Inc.) display a checkbook register with labeled columns for check number, date, payee, and amount. Changes can be made in place, new entries can be made at the first blank line, and a check-mark can be made to indicate verification against a monthly report or bank statement. Users can search for a particular payee by filling in a blank payee field and then typing a `?`.

Why not have web-based airline-reservations systems show the user a map and prompt for cursor motion to the departing and arriving cities? Then the user can select the date from a calendar and a time from a clock. Showing the seating plan of the plane on the screen, with a diagonal line to indicate an already-reserved seat, would enable seat selection.

"Direct manipulation" is an accurate description of the programming of certain industrial robot tools. The operator holds the robot "hand" and guides it through a spray painting or welding task while the controlling computer records every action. The control computer can then operate the robot automatically, repeating the precise action as many times as is necessary.

Why not teach students about polynomial equations by letting them move sliders to set values for the coefficients and watch how the graph changes, where the y-axis intercept occurs, or how the derivative equation reacts. Similarly, direct manipulation of sliders for red, green, and blue is a satisfying way to explore color space. Slider-based dynamic queries are a powerful tool for information exploration (see Section 15.4).

Direct manipulation has the power to attract users because it is comprehensible, rapid, and even enjoyable. If actions are simple, reversibility is ensured, and retention is easy, then anxiety recedes, users feel in control, and satisfaction flows in.

6.3 Explanations of Direct Manipulation

Several authors have attempted to describe the component principles of direct manipulation. An imaginative and early interactive system designer, Ted Nelson (1980), perceived user excitement when the interface was constructed by what he calls the *principle of virtuality*—a representation of reality that can be manipulated. Rutkowski (1982) conveyed a similar concept in his *principle of transparency*: "The user is able to apply intellect directly to the task; the tool itself seems to disappear." Heckel (1991) laments that "Our

instincts and training as engineers encourage us to think logically instead of visually, and this is counterproductive to friendly design." His description is in harmony with the popular notions of logical symbolic sequential left-brain and the visual artistic all-at-once right-brain problem-solving styles.

Hutchins et al. (1986) review the concepts of direct manipulation and offer a thoughtful decomposition of concerns. They describe the "feeling of involvement directly with a world of objects rather than of communicating with an intermediary," and clarify how direct manipulation breaches the *gulf of execution* and the *gulf of evaluation*.

These writers and others (Ziegler and Fahnrich, 1988; Thimbleby, 1990; Phillips and Apperley, 1991; Frohlich, 1993) support the recognition that a new form of interactive system had emerged. Much credit also goes to the individual designers who created systems that exemplify aspects of direct manipulation.

Another perspective on direct manipulation comes from the psychology literature on *problem-solving* and *learning research*. Suitable representations of problems have been clearly shown to be critical to solution finding and to learning. Polya (1957) suggests drawing a picture to represent mathematical problems. This approach is in harmony with Maria Montessori's teaching methods for children (Montessori, 1964). She proposed use of physical objects, such as beads or wooden sticks, to convey such mathematical principles as addition, multiplication, or size comparison. The durable abacus is appealing because it gives a direct-manipulation representation of numbers.

Bruner (1966) extended the physical-representation idea to cover polynomial factoring and other mathematical principles. Carroll, Thomas, and Malhotra (1980) found that subjects given spatial representation were faster and more successful in problem solving than were subjects given an isomorphic problem with a temporal representation. Similarly, Te'eni (1990) found that the feedback in direct-manipulation designs was effective in reducing users' logical errors in a task requiring statistical analysis of student grades. The advantage appears to stem from having the data entry and display combined in a single location on the display.

Physical, spatial, or visual representations also appear to be easier to retain and manipulate than are textual or numeric representations (Arnheim, 1972; McKim, 1980). Wertheimer (1959) found that subjects who memorized the formula for the area of a parallelogram, $A = h \times b$, rapidly succeeded in doing such calculations. On the other hand, subjects who were given the structural understanding of cutting off a triangle from one end and placing it on the other end could more effectively retain the knowledge and generalize it to solve related problems. In plane-geometry theorem proving, spatial representation facilitates discovery of proof procedures over a strictly axiomatic representation of Euclidean geometry. The diagram provides heuristics that are difficult to extract from the axioms. Similarly, students are often encouraged to solve algebraic word problems by drawing pictures to represent those problems.

Papert's (1980) LOGO language created a mathematical microworld in which the principles of geometry are visible. Based on the Swiss psychologist Jean Piaget's theory of child development, LOGO offers students the opportunity to create line drawings easily with an electronic turtle displayed on a screen. In this environment, users derive rapid feedback about their programs, can determine what has happened easily, can spot and repair errors quickly, and can gain satisfaction from creative production of drawings. These features are all characteristic of a direct-manipulation environment.

6.3.1 Problems with direct manipulation

Spatial or visual representations are not necessarily an improvement over text, because they may be too spread out, causing off-page connectors on paper or tedious scrolling on displays. In professional programming, use of high-level flowcharts and database-schema diagrams can be helpful for some tasks, but there is a danger that they will be confusing. Similarly, direct-manipulation designs may consume valuable screen space and thus force valuable information offscreen, requiring scrolling or multiple actions. Studies of graphical plots versus tabular business data and of flowcharts versus program text demonstrate advantages for graphical approaches when pattern-recognition tasks are relevant, but disadvantages when the graphic gets too large and the tasks require detailed information. For experienced users, a tabular textual display of 50 document names may be more appropriate than only 10 graphic document icons with the names abbreviated to fit the icon size.

A second problem is that users must learn the meaning of components of the visual representation. A graphic icon may be meaningful to the designer, but may require as much or more learning time than a word. Some airports that serve multilingual communities use graphic icons extensively, but the meanings of these icons may not be obvious. Similarly, some computer terminals designed for international use have icons in place of names, but the meaning is not always clear. Icons with titles that appear when the cursor is over them offer only a partial solution.

A third problem is that the visual representation may be misleading. Users may grasp the analogical representation rapidly, but then may draw incorrect conclusions about permissible actions. Users may overestimate or underestimate the functions of the computer-based analogy. Ample testing must be carried out to refine the displayed objects and actions and to minimize negative side effects.

A fourth problem is that, for experienced typists, taking your hand off the keyboard to move a mouse or point with a finger may be slower than typing the relevant command. This problem is especially likely to occur if the user is familiar with a compact notation, such as arithmetic expressions, that is easy

to enter from a keyboard, but that may be more difficult to select with a mouse. The keyboard remains the most effective direct-manipulation device for certain tasks.

Choosing the right objects and actions is not necessarily an easy task. Simple metaphors, analogies, or models with a minimal set of concepts are a good starting point. Mixing metaphors from two sources may add complexity that contributes to confusion. The emotional tone of the metaphor should be inviting rather than distasteful or inappropriate (Carroll and Thomas, 1982)—sewage-disposal systems are an inappropriate metaphor for electronic-message systems. Since the users may not share the metaphor, analogy, or conceptual model with the designer, ample testing is required. For help in training, an explicit statement of the model, of the assumptions, and of the limitations is necessary.

6.3.2 The OAI model explanation of direct manipulation

The attraction of direct manipulation is apparent in the enthusiasm of the users. The designers of the examples in Section 6.2 had an innovative inspiration and an intuitive grasp of what users would want. Each example has features that we could criticize, but it will be more productive for us to construct an integrated portrait of direct manipulation with three principles:

1. Continuous representation of the objects and actions of interest with meaningful visual metaphors
2. Physical actions or presses of labeled buttons, instead of complex syntax
3. Rapid incremental reversible operations whose effect on the object of interest is visible immediately

Using these three principles, it is possible to design systems that have these beneficial attributes:

- Novices can learn basic functionality quickly, usually through a demonstration by a more experienced user.
- Experts can work rapidly to carry out a wide range of tasks, even defining new functions and features.
- Knowledgeable intermittent users can retain operational concepts.
- Error messages are rarely needed.
- Users can immediately see whether their actions are furthering their goals, and, if the actions are counterproductive, they can simply change the direction of their activity.
- Users experience less anxiety because the system is comprehensible and because actions can be reversed easily.

- Users gain confidence and mastery because they are the initiators of action, they feel in control, and they can predict the system responses.

The success of direct manipulation is understandable in the context of the OAI model. The object of interest is displayed so that interface actions are close to the high-level task domain. There is little need for the mental decomposition of tasks into multiple interface commands with a complex syntactic form. On the contrary, each action produces a comprehensible result in the task domain that is visible in the interface immediately. The closeness of the task domain to the interface domain reduces operator problem-solving load and stress. This basic principle is related to stimulus-response compatibility, as discussed in the human-factors literature. The task objects and actions dominate the users' concerns, and the distraction of dealing with a tedious interface is reduced (Fig. 6.10).

In contrast to textual descriptors, dealing with visual representations of objects may be more "natural" and closer to innate human capabilities: Action and visual skills emerged well before language in human evolution. Psychologists have long known that people grasp spatial relation-

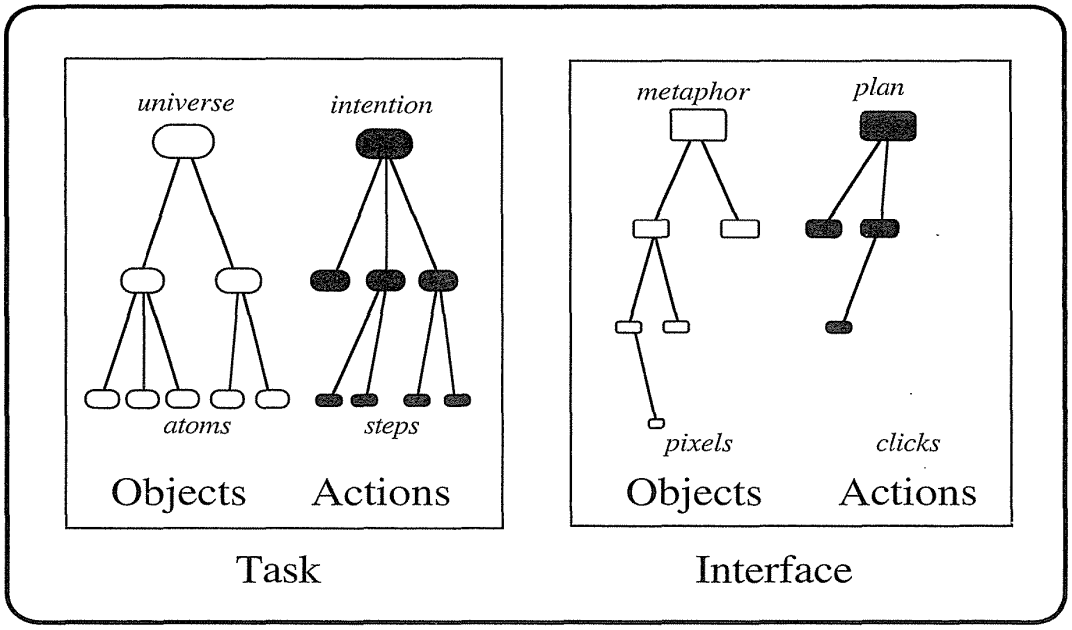


Figure 6.10

Direct-manipulation systems may require users to learn substantial task knowledge. However, users must acquire only a modest amount of interface knowledge and syntactic details.

ships and actions more quickly when those people are given visual rather than linguistic representations. Furthermore, intuition and discovery are often promoted by suitable visual representations of formal mathematical systems.

The Swiss psychologist Jean Piaget described *four stages of development: sensorimotor* (from birth to approximately 2 years), *preoperational* (2 to 7 years), *concrete operational* (7 to 11 years), and *formal operations* (begins at approximately 11 years) (Copeland, 1979). According to this theory, physical actions on an object are comprehensible during the concrete operational stage, and children acquire the concept of *conservation* or *invariance*. At about age 11, children enter the formal-operations stage, in which they use *symbol manipulation* to represent actions on objects. Since mathematics and programming require abstract thinking, they are difficult for children, and designers must link symbolic representations to actual objects. Direct manipulation brings activity to the concrete-operational stage, thus making certain tasks easier for children and adults.

6.4 Visual Thinking and Icons

The concepts of a *visual language* and of *visual thinking* were promoted by Arnheim (1972), and were embraced by commercial graphic designers (Verplank, 1988; Mullet and Sano, 1995), semiotically oriented academics (*semiotics* is the study of signs and symbols), and data-visualization gurus. The computer provides a remarkable visual environment for revealing structure, showing relationships, and enabling interactivity that attracts users who have artistic, right-brained, holistic, intuitive personalities. The increasingly visual nature of computer interfaces can sometimes challenge or even threaten the logical, linear, text-oriented, left-brained, compulsive, rational programmers who were the heart of the first generation of hackers. Although these stereotypes—or caricatures—will not stand up to scientific analysis, they do convey the dual paths that computing is following. The new visual directions are sometimes scorned by the traditionalists as *WIMP* (windows, icons, mouse, and pull-down menu) interfaces, whereas the command-line devotees are seen as inflexible, or even stubborn.

There is evidence that different people have different cognitive styles, and it is quite understandable that individual preferences may vary. Just as there are multiple ice-cream flavors or car models, so too there will be multiple interface styles. It may be that preferences will vary by user and by tasks. So respect is due to each community, and the designer's goal is to provide the best of each style and the means to cross over when desired.

The conflict between text and graphics becomes most heated when the issue of *icons* is raised. Maybe it is not surprising that the dictionary definitions of *icon* usually refer to religious images, but the central notion in computing is that an icon is an image, picture, or symbol representing a concept (Rogers, 1989; Marcus, 1992). In the computer world, icons are usually small (less than 1-inch-square or 64- by 64-pixel) representations of an object or action. Smaller icons are often used to save space or to be integrated within other objects, such as a window border or toolbar. It is not surprising that icons are often used in painting programs to represent the tools or actions (lasso or scissors to cut out an image, brush for painting, pencil for drawing, eraser to wipe clean), whereas word processors usually have textual menus for their actions. This difference appears to reflect the differing cognitive styles of visually and textually oriented users, or at least differences in the tasks. Maybe, while you are working on visually oriented tasks, it is helpful to "stay visual" by using icons, whereas, while you are working on a text document, it is helpful to "stay textual" by using textual menus.

For situations where both a visual icon or a textual item are possible—for example, in a directory listing—designers face two interwoven issues: how to decide between icons and text, and how to design icons. The well-established highway signs are a useful source of experience. Icons are unbeatable for showing ideas such as a road curve, but sometimes a phrase such as ONE WAY!—DO NOT ENTER is more comprehensible than an icon. Of course, the smorgasbord approach is to have a little of each (as with, for example, the octagonal STOP sign), and there is evidence that icons plus words are effective in computing situations (Norman, 1991). So the answer to the first question (deciding between icons and text) depends not only on the users and the tasks, but also on the quality of the icons or the words that are proposed. Textual menu choices are covered in Chapter 7; many of the principles carry over. In addition, these icon-specific guidelines should be considered:

- Represent the object or action in a familiar and recognizable manner.
- Limit the number of different icons.
- Make the icon stand out from its background.
- Consider three-dimensional icons; they are eye catching, but also can be distracting.
- Ensure that a single selected icon is clearly visible when surrounded by unselected icons.
- Make each icon distinctive from every other icon.
- Ensure the harmoniousness of each icon as a member of a family of icons.

- Design the movement animation: when dragging an icon, the user might move the whole icon, just a frame, possibly a grayed-out or transparent version, or a black box.
- Add detailed information, such as shading to show size of a file (larger shadow indicates larger file), thickness to show breadth of a directory folder (thicker means more files inside), color to show the age of a document (older might be yellower or grayer), or animation to show how much of a document has been printed (a document folder is absorbed progressively into the printer icon).
- Explore the use of combinations of icons to create new objects or actions—for example, dragging a document icon to a folder, trashcan, outbox, or printer icon has great utility. Can a document be appended or prepended to another document by pasting of adjacent icons? Can a user set security levels by dragging a document or folder to a guard dog, police car, or vault icon? Can two database icons be intersected by overlapping of the icons?

Marcus (1992) applies semiotics as a guide to four levels of icon design:

1. *Lexical qualities* Machine-generated marks—pixel shape, color, brightness, blinking
2. *Syntactics* Appearance and movement—lines, patterns, modular parts, size, shape
3. *Semantics* Objects represented—concrete versus abstract, part versus whole
4. *Pragmatics* Overall legible, utility, identifiable, memorable, pleasing

He recommends starting by creating quick sketches, pushing for consistent style, designing a layout grid, simplifying appearance, and evaluating the designs by testing with users. We might consider a fifth level of icon design:

5. *Dynamics* Receptivity to clicks—highlighting, dragging, combining

The dynamics of icons might also include a rich set of gestures with a mouse, touchscreen, or pen. The gestures might indicate copy (up and down), delete (a cross), edit (a circle), and so on. Icons might also have associated sounds. For example, if each document icon had associated with it a tone (the lower the tone, the bigger the document), then, when a directory was opened, each tone might be played simultaneously or sequentially. Users might get used to the symphony played by each directory and could detect certain features or anomalies, just as we often know telephone numbers by tune and can detect misdialings as discordant tones.

Icon design becomes more interesting as computer hardware improves and as designers become more creative. Animated icons that demonstrate

their function improve online help capabilities (see Section 12.4.2). Beyond simple icons, we are now seeing increasing numbers of visual programming languages (see Section 5.3.1) and specialized languages for mechanical engineering, circuit design, and database query.

6.5 Direct-Manipulation Programming

Performing tasks by direct manipulation is not the only goal. It should be possible to do programming by direct manipulation as well, at least for certain problems. People sometimes program robots by moving the robot arm through a sequence of steps that are later replayed, possibly at higher speed. This example seems to be a good candidate for generalization. How about moving a drill press or a surgical tool through a complex series of motions that are then repeated exactly? In fact, these direct-manipulation–programming ideas are implemented in modest ways with automobile radios that users preset by tuning to their desired station and then pressing and holding a button. Later, when the button is pressed, the radio tunes to the preset frequency. Some professional television-camera supports allow the operator to program a sequence of pans or zooms and then to replay it smoothly when required.

Programming of physical devices by direct manipulation seems quite natural, and an adequate visual representation of information may make direct-manipulation programming possible in other domains. Several word processors allow users to create macros by simply performing a sequence of commands and storing it for later use. WordPerfect enables the creation of macros that are sequences of text, special function keys such as TAB, and other WordPerfect commands. emacs allows its rich set of functions, including regular expression searching, to be recorded into macros. Macros can invoke one another, leading to complex programming possibilities. These and other systems allow users to create programs with nonvarying action sequences using direct manipulation, but strategies for including loops and conditionals vary. emacs allows macros to be encased in a loop with simple repeat factors. emacs and WordPerfect also allow users to attach more general control structures by resorting to textual programming languages.

Spreadsheet packages, such as LOTUS 1-2-3 and Excel, have rich programming languages and allow users to create portions of programs by carrying out standard spreadsheet operations. The result of the operations is stored in another part of the spreadsheet and can be edited, printed, and stored in a textual form.

Macro facilities in GUIs are more challenging to design than are macro facilities in traditional command interfaces. The MACRO command of Direct

Manipulation Disk Operating System (DMDOS) (Iseki and Shneiderman, 1986) was an early attempt to support a limited form of programming for file movement, copying, and directory commands.

Smith (1977) inspired work in this area with his Pygmalion system that allowed arithmetic programs to be specified visually with icons. A number of early research projects have attempted to create direct-manipulation programming systems (Rubin et al., 1985). Maulsby and Witten (1989) developed a system that could induce or infer a program from examples, questioning the users to resolve ambiguities. In constrained domains, inferences become predictable and useful, but if the inference is occasionally wrong, users will quickly distrust it.

Myers (1992) coined the phrase *demonstrational programming* to characterize the technique of letting users create macros by simply doing their tasks and having the system construct the proper generalization automatically. Cypher (1991) built and ran a usability test with seven subjects for his EAGER system that monitored user actions within HyperCard. When EAGER recognized two similar sequences, a small smiling cat appeared on the screen to offer the users help in carrying out further iterations. Cypher's success with two specific tasks is encouraging, but it has proved to be difficult to generalize this approach.

It would be helpful if the computer could recognize repeated patterns reliably and create useful macros automatically, while the user was engaged in performing a repetitive interface task. Then, with the user's confirmation, the computer could take over and could carry out the remainder of the task automatically. This hope for automatic programming is appealing, but a more effective approach may be to give users the visual tools to specify and record their intentions. Rule-based programming with graphical conditions and actions offers a fresh alternative that may be appealing to children and adults (Fig. 6.11) (Smith et al., 1994). The screen is portrayed as a set of tiles, and users specify graphical rewrite rules by showing before-and-after tile examples. Another innovative environment conceived of initially for children is ToonTalk (Kahn, 1996), which offers animated cartoon characters who carry out actions in buildings using a variety of fanciful tools.

To create a reliable tool that works in many situations without unpredictable automatic programming, designers must meet the *five challenges of programming in the user interface (PITUI)* (Potter, 1993):

1. Sufficient computational generality (conditionals, iteration)
2. Access to the appropriate data structures (file structures for directories, structural representations of graphical objects) and operators (selectors, booleans, specialized operators of applications)
3. Ease in programming (by specification, by example, or by demonstration, with modularity, argument passing, and so on) and in editing programs

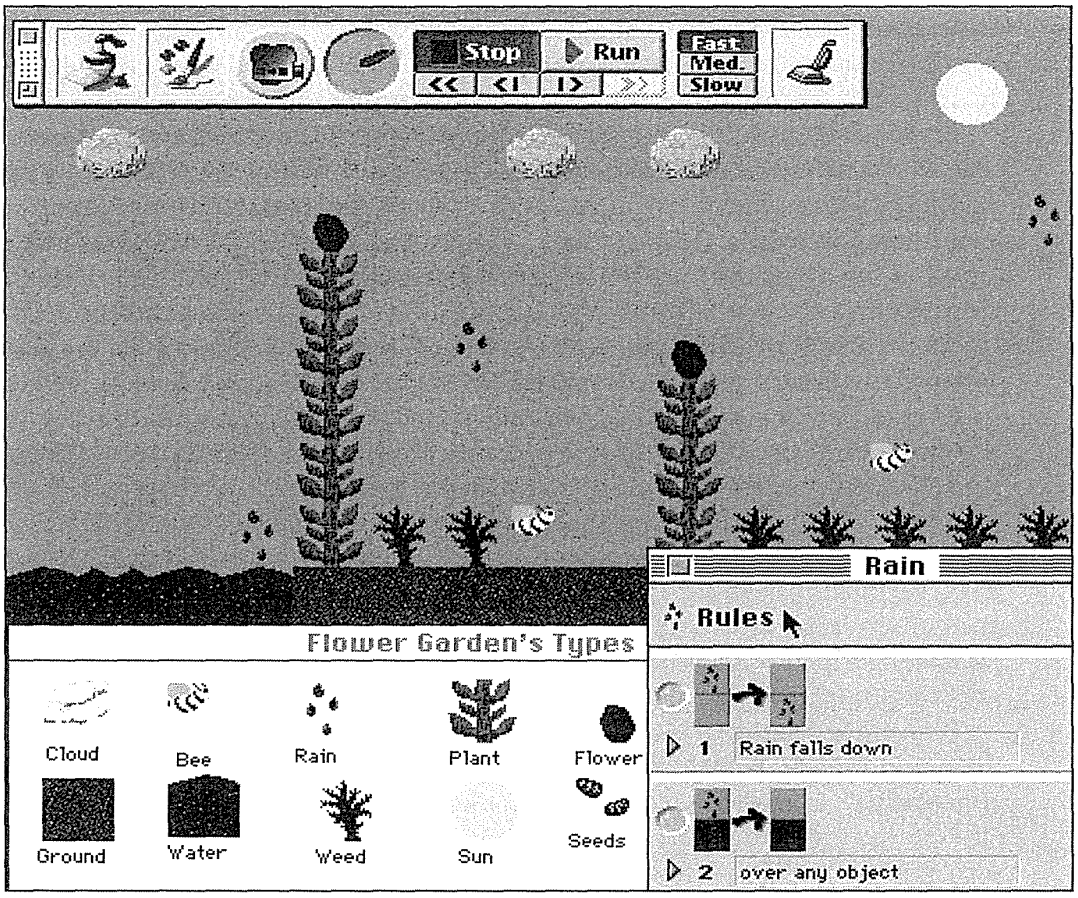


Figure 6.11

Cocoa display showing the Flower Garden world, with the control panel, the garden data types, and the graphical rules for the rain falling down and getting absorbed by any object. (Used with permission of Apple Computers, Inc., Cupertino, CA.)

4. Simplicity in invocation and assignment of arguments (direct manipulation, simple library strategies with meaningful names or icons, in-context execution, and availability of results)
5. Low risk (high probability of bug-free programs, halt and resume facilities to permit partial executions, undo operations to enable repair of unanticipated damage)

The goal of PITUI is to allow users easily and reliably to repeat automatically the actions that they can perform manually in the user interface. Rather than depending on unpredictable inferencing, users will be able to indicate their intentions explicitly by manipulating objects and actions. The design of

direct-manipulation systems will undoubtedly be influenced by the need to support PITUI. This influence will be a positive step that will also facilitate history keeping, undo, and online help.

The *cognitive-dimensions framework* may help us to analyze design issues of visual-programming environments, such as those needed for PITUI (Green and Petre, 1996). The framework provides a vocabulary to facilitate discussion of high-level design issues; for example, *viscosity* is used to describe the difficulty of making changes in a program, and *progressive evaluation* describes the capacity for execution of partial programs. Other dimensions are consistency, diffuseness, hidden dependencies, premature commitment, and visibility.

Direct-manipulation programming offers an alternative to the agent scenarios (see Section 2.9). Agent promoters believe that the computer can ascertain the users' intentions automatically, or can take action based on a vague statements of goals. I doubt that user intentions are so easily determined or that vague statements are usually effective. However, if users can specify what they want with comprehensible actions selected from a visual display, then they can often and rapidly accomplish their goals while preserving their sense of control and accomplishment.

6.6 Home Automation

Internationally, many companies predict a large market in extensive controls in homes, but only if the user interfaces can be made simple. Remote control of devices (either from one part of the home to another, from outside, or by programmed delays) is being extended to channel audio and video throughout the house, to schedule lawn watering as a function of ground moisture, to offer video surveillance and burglar alarms, and to provide multiple-zone environmental controls plus detailed maintenance records.

Some designers promote voice controls, but commercially successful systems use traditional pushbuttons, remote controllers, telephone keypads, and touchscreens, with the latter proving to be the most popular. Installations with two to 10 touchscreens spread around the house should satisfy most homeowners. Providing direct-manipulation controls with rich feedback is vital in these applications. Users are willing to take training, but operation must be rapid and easy to remember even if the option is used only once or twice per year (such as spring and fall adjustments for daylight-savings time).

Studies of four touchscreen designs, all based on direct manipulation, explored scheduling operations for VCR recording and light controls (Plaisant et al., 1990; Plaisant and Shneiderman, 1991). The four designs were

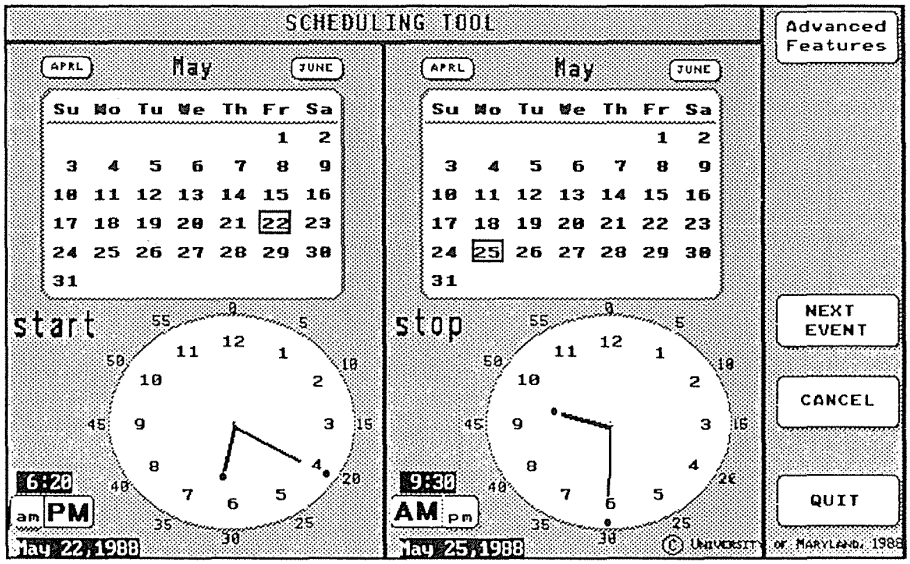


Figure 6.12

This scheduler shows two calendars for start and stop dates, plus two 12-hour circular clocks with hands that the user can drag to set start and stop times. (Used with permission of University of Maryland, College Park, MD.)

1. A digital clock that users set by pressing step keys (similar to onscreen programming in current videocassette players)
2. A 24-hour circular clock whose hands users can drag with fingers
3. A 12-hour circular clock (plus A.M.–P.M. toggle) whose hands users can drag with fingers (Fig. 6.12)
4. A 24-hour time line in which ON–OFF flags can be placed to indicate start-stop times (Fig. 6.13)

The results indicated that the 24-hour time line was easiest to understand and use. Direct-manipulation principles were central to this design; users selected dates by touching a monthly calendar, and times by moving the ON or OFF flags on to the 24-hour time line. The flags were an effective way of representing the ON or OFF actions and of specifying times without use of a keyboard. The capacity to adjust the flag locations incrementally, and the ease of removing them, were additional benefits. We are extending the design to accommodate more complex tasks, such as scheduling and synchronization of multiple devices, searching through schedules to find dates with specific events, scheduling repeated events (close curtains every night at dusk, turn lights on every Friday night at 7 P.M., record status monthly),

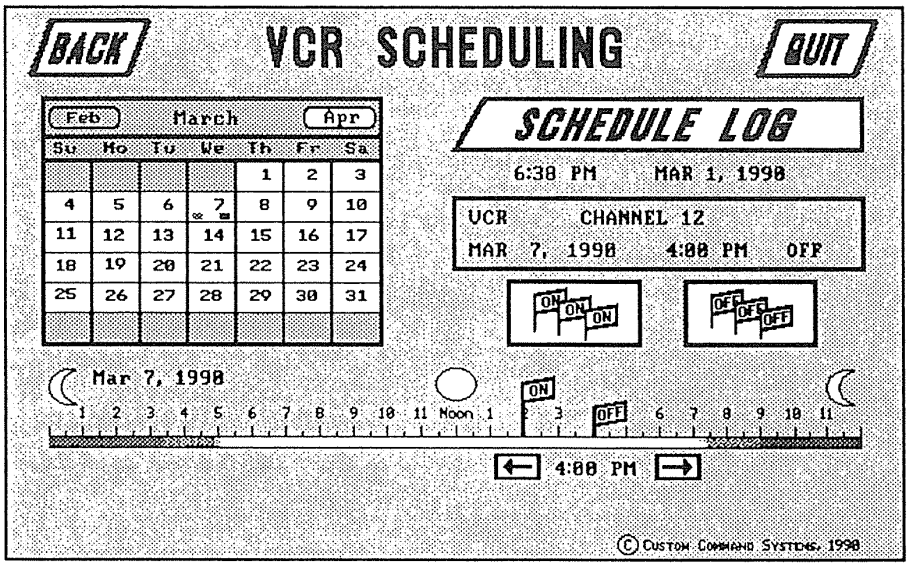


Figure 6.13

This 24-hour time-line scheduler was most successful in our usability studies. The users select a date by pointing on the calendar and then dragging ON and OFF flags to the 24-hour time lines. The feedback is a red line on the calendar and the time lines. (Used with permission of University of Maryland, College Park, MD.)

and long-duration events. A generalization of the flags-on-a-line idea was applied to heating control, where users specified upper and lower bounds by dragging flags on a thermometer.

Since so much of home control involves the room layouts and floorplans, many direct-manipulation actions take place on a display of the floorplan (Fig. 6.14), with selectable icons for each status indicator (such as burglar alarm, heat sensor, or smoke detector), and for each activator (such as curtain or shade closing and opening motors, airconditioning- or heating-vent controllers, or audio and video speaker or screen). People could route sound from a CD player located in the living room to the bedroom and kitchen by merely dragging the CD icon into those rooms. Sound-volume control would be accomplished by having the user move a marker on a linear scale.

The simple act of turning a device ON or OFF proved to be an interesting problem. Wall-mounted light switches typically show their status by up for ON and down for OFF. Most people have learned this standard and can get what they want on the first try, if they know which switch to throw to turn on a specific light. Laying out the switches to reflect the floorplan does solve

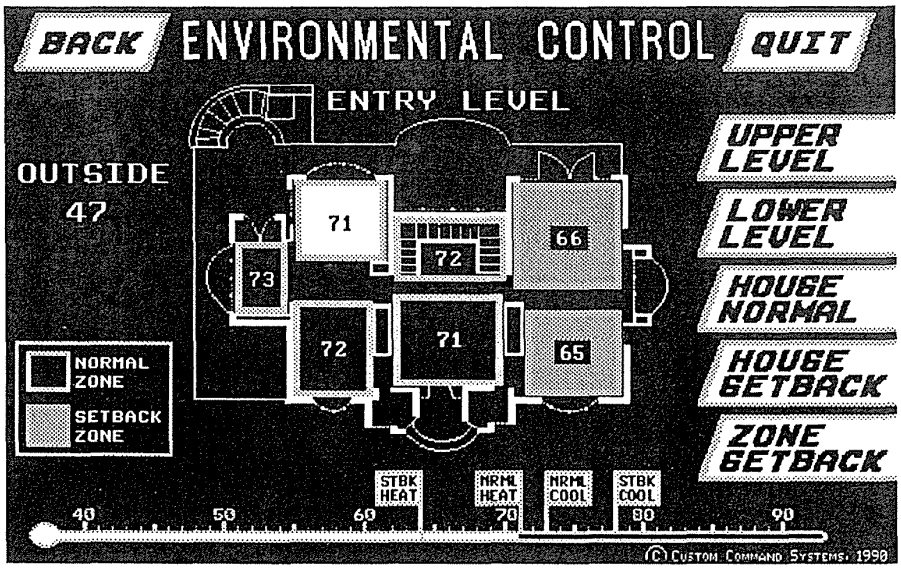


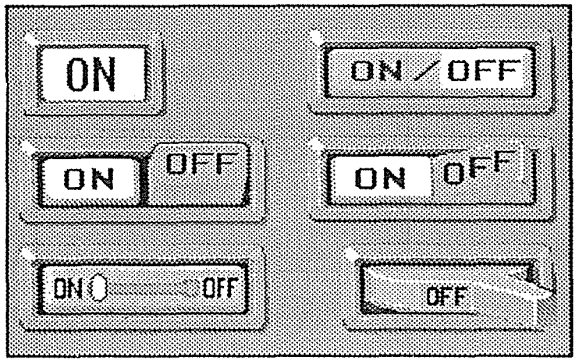
Figure 6.14

Floorplan of a private home, used to set temperatures. Direct-manipulation designs emphasize task-domain graphics. (Courtesy of Custom Command Systems, College Park, MD.)

the problem nicely (Norman, 1988). Visitors may have problems because, in some countries, ON and OFF are reversed or the up-down switches have been replaced by push buttons. To explore possibilities, we constructed six kinds of touchscreen ON-OFF buttons with three-dimensional animation and sound (Fig. 6.15). There were significant differences in user preferences, with high marks going to the simple button, the rocker, and multiple-level pushbuttons. The multiple pushbuttons have a readily comprehensible

Figure 6.15

Varying designs for toggle buttons using three-dimensional graphic characteristics. Designed by Catherine Plaisant.



visual presentation, and they generalize nicely to multiple state devices (OFF, LOW, MEDIUM, HIGH).

Controlling complex home equipment from a touchscreen by direct manipulation reshapes how we think of homes and their residents. New questions arise, such as whether residents will feel safer, be happier, save more money, or experience more relaxation with these devices. Are there new notations, such as petri-net variants or role-task diagrams, for describing home automation and the social relations among residents? The benefits to users who have disabilities or are elderly were often on our minds as we designed these systems, since these people may be substantial beneficiaries of this technology, even though initial implementations are designed for the healthy and wealthy.

6.7 Remote Direct Manipulation

There are great opportunities for the teleoperation or remote control of devices if acceptable user interfaces can be constructed. If designers can provide adequate feedback in sufficient time to permit effective decision making, then attractive applications in office automation, computer-supported collaborative work, education, and information services may become viable. Remote-controlled environments in medicine could enable specialists to provide consultations more rapidly, or allow surgeons to conduct more complex procedures during operations. Home-automation applications could extend remote operation of telephone-answering machines to security and access systems, energy control, and operation of appliances. Scientific applications in space, underwater, or in hostile environments can enable new research projects to be conducted economically and safely (Uttal, 1989; Sheridan, 1992).

In traditional direct-manipulation systems, the objects and actions of interest are shown continuously; users generally point, click, or drag, rather than type; and feedback, indicating change, is immediate. However, when the devices being operated are remote, these goals may not be realizable, and designers must expend additional effort to help users to cope with slower response, incomplete feedback, increased likelihood of breakdowns, and more complex error recovery. The problems are strongly connected to the hardware, physical environment, network design, and the task domain.

A typical remote application is *telemedicine*: medical care delivered over communication links (Satava and Jones, 1996). In one scenario, the physician specialist being consulted and the patient's primary physician or a technician are in different locations. Then, for example, an effective telepathology

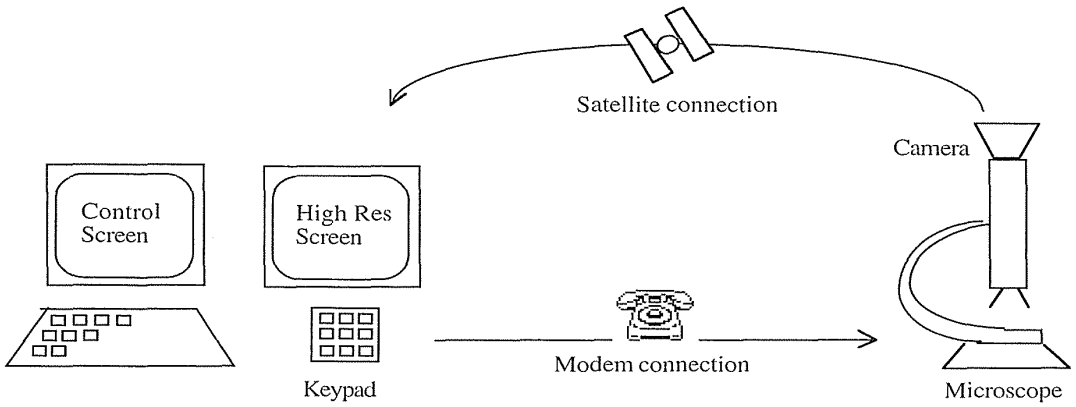


Figure 6.16

A simplified diagram of a telepathology system showing control actions sent by telephone and images sent by satellite.

system (Weinstein et al., 1989) allows a pathologist to examine tissue samples or body fluids under a remotely located microscope (Figs. 6.16 and 6.17). The transmitting workstation has a high-resolution camera mounted on a motorized light microscope. The image is transmitted via broadband satellite, microwave, or cable. The consulting pathologist at the receiving workstation can manipulate the microscope using a keypad, and can see a high-resolution image of the magnified sample. The two care givers talk by telephone to coordinate control and to request slides that are placed manually under the microscope. Controls include

- Magnification (three or six objectives)
- Focus (coarse and fine bidirectional control)
- Illumination (bidirectional adjustment continuous or by step)
- Position (two-dimensional placement of the slide under the microscope objective)

The architecture of remote environments introduces several complicating factors:

- *Time delays* The network hardware and software cause delays in sending user actions and receiving feedback: a *transmission delay,* or the time it takes for the command to reach the microscope (in our example,

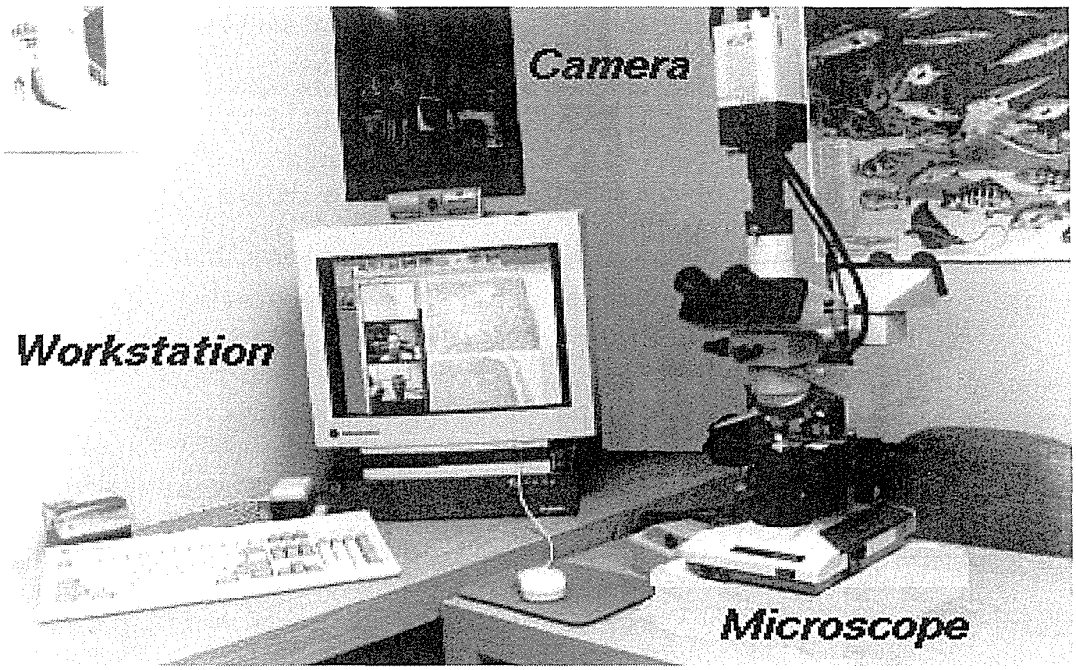


Figure 6.17

Telepathology components include a microscope with a camera attached to a workstation. This setup enables a pathologist to use remote control to examine the slides. (Used with permission of William J. Chimiak and Robert O. Rainer, The Bowman Gray School of Medicine of Wake Forest University, Winston Salem, NC.)

transmitting the command through the modem), and *operation delay*, or the time until the microscope responds (Van de Vegte et al., 1990). These delays in the system prevent the operator from knowing the current status of the system. For example, if a positioning command has been issued, it may take several seconds for the slide to start moving. As the feedback appears showing the motion, the users may recognize that they are going to overshoot their destination, but a few seconds will pass before the stopping command takes effect.

- *Incomplete feedback* Devices originally designed for direct control may not have adequate sensors or status indicators. For instance, the microscope can transmit its current position, but it operates so slowly that it cannot be used continuously. Thus, it is not possible to indicate on the control screen the exact current position relative to the start and desired positions.
- *Feedback from multiple sources* Incomplete feedback is different from no feedback. The image received on the high-resolution screen is the

main feedback to evaluate the result of an action. In addition, the microscope can occasionally report its exact position, allowing recalibration of the status display. It is also possible to indicate the estimated stage position during the execution of a movement. This estimated feedback can be used as a progress indicator whose accuracy depends on the variability of the time delays. To comply with the physical incompatibility between the high-resolution feedback (analog image) and the rest of the system (digital), we spread the multiple feedbacks over several screens. Thus, the pathologists are forced to switch back and forth between multiple sources of feedback, increasing their cognitive load.

- *Unanticipated interferences* Since the devices operated are remote, and may be also operated by other persons in this or another remote location, unanticipated interferences are more likely to occur than in traditional direct-manipulation environments. For instance, if the slide under the microscope were moved (accidentally) by a local operator, the positions indicated might not be correct. A breakdown might also occur during the execution of a remote operation, without a good indication of this event being sent to the remote site. Such breakdowns require increased status information for remote users and additional actions that allow for correction.

One solution to these problems is to make explicit the network delays and breakdowns as part of the system. The user sees a model of the starting state of the system, the action that has been initiated, and the current state of the system as it carries out the action. It may be preferable to provide spatially parameterized positioning actions (for example, move by a distance $+x$, $+y$, or move to a fixed point (x, y) in a two-dimensional space), rather than providing temporal commands (for example, start moving right at a 36° angle from the horizontal). In other words, the users specify a destination (rather than a motion), and wait until the action is completed before readjusting the destination if necessary.

Remote direct manipulation is rooted in two domains that, so far, have been independent. The first root grows from direct manipulation in personal computers and is often identified with the desktop metaphor and office automation. The second root is in process control, where human operators control physical processes in complex environments. Typical tasks are operating power or chemical plants, controlling manufacturing, flying airplanes, or steering vehicles. If the physical processes take place in a remote location, we talk about *teleoperation* or *remote control*. To perform the control task, the human operator may interact with a computer, which may carry out some of the control tasks without any interference by the human operator. This idea is captured by the notion of *supervisory control* (Sheridan, 1992). Although

supervisory control and direct manipulation stem from different problem domains and are usually applied to different system architectures, they carry a strong resemblance.

6.8 Virtual Environments

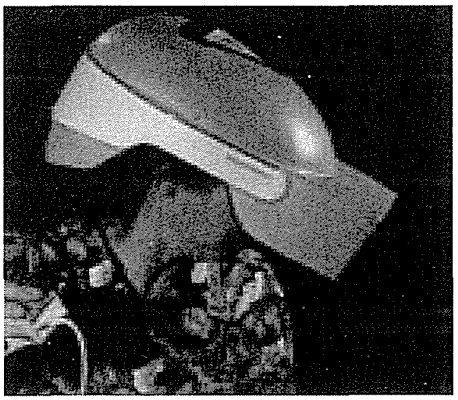
Flight-simulator designers use many tricks to create the most realistic experience for fighter or airline pilots. The cockpit displays and controls are taken from the same production line that create the real ones. Then, the windows are replaced by high-resolution computer displays, and sounds are choreographed to give the impression of engine start or reverse thrust. Finally, the vibration and tilting during climbing or turning are created by hydraulic jacks and intricate suspension systems. This elaborate technology may cost almost $100 million, but even then it is a lot cheaper, safer, and more useful for training than the $400-million jet that it simulates. Of course, home videogame players have purchased millions of $30 flight simulators that run on their personal computers. Flying a plane is a complicated and specialized skill, but simulators are available for more common—and for some surprising—tasks under the alluring name of *virtual reality* or the more descriptive *virtual environments*.

High above the office desktop, much beyond multimedia, and farther out than the hype of hypermedia, the gurus and purveyors of virtuality are promoting immersive experiences (Fig. 6.18). Whether soaring over Seattle, bending around bronchial tubes to find lung cancers, or grasping complex molecules, the cyberspace explorers are moving past their initial fantasies to create useful technologies. The imagery and personalities involved in virtual reality are often colorful (Rheingold, 1991), but many researchers have tried to present a balanced view by conveying enthusiasm while reporting on problems (MacDonald and Vince, 1994; Bryson, 1996).

Architects have been using computers to draw three-dimensional representations of buildings for two decades. Most of their design systems show the building on a standard or slightly larger display, but adding a large-screen projector to create a wall-sized image gives prospective clients a more realistic impression. Now add animation that allows clients to see what happens if they move left or right, or approach the image. Then enable clients to control the animation by walking on a treadmill (faster walking brings the building closer more quickly), and allow them to walk through the doors or up the stairs. Finally, replace the large-screen projector with a head-mounted display, and monitor head movement with Polhemus trackers. Each change

Figure 6.18

In the goggles-and-gloves approach to virtual reality, the system tracks the user's hand and head motions, plus finger gestures, to control the scene's movement and manipulation. To enter this virtual environment you need special gear. Any of several types of stereoscopic devices transform otherwise two-dimensional image data into three-dimensional images. Some three-dimensional viewers, called head-mounted displays, resemble helmets with movie screens where the visor would be. (NCSA/University of Illinois.)



takes users a bit farther along the range from "looking at" to "being in." Bumping into walls, falling (gently) down stairs, meeting other people, or having to wait for an elevator could be the next variations.

The architectural application is a persuasive argument for "being in," because we are used to "being in" buildings and moving around them. On the other hand, for many applications, "looking at" is often more effective, which is why air-traffic–control workstations place the viewer above the situation display. Similarly, seeing movies on the large wraparound screens that put viewers "in" race cars or airplanes are special events compared to the more common "looking at" television experience. The Living Theater of the 1960s created an involving theatrical experience and "be-ins" were popu-

lar, but most theater goers prefer to take their "suspension of disbelief" experiences from the "looking at" perspective (Laurel, 1991).

It remains to be seen whether doctors, accustomed to "looking at" a patient, really want to crawl through the patient's lungs or "be in" the patient's brains. Modern surgical procedures and technology can benefit by "looking at" video images from inside a patient's heart taken through fiber-optic cameras and from use of remote direct-manipulation devices that minimize the invasive surgery. Surgery planning can also be done with three-dimensional "looking at" visualizations shown on a traditional desktop display and guided by hand-held props (Hinckley et al., 1994). There are more mundane applications for such video and fiberoptic magic; imagine the benefits to household plumbers of being able to see lost wedding rings around the bends of a sink drain or to see and grasp the child's toy that has fallen down the pipes of a now-clogged toilet.

Other concepts that were sources for the current excitement include *artificial reality*, pioneered by Myron Krueger (1991). His VideoPlace and VideoDesk installations with large-screen projectors and video sensors combined full-body movement with projected images of light creatures that walked along a performer's arm or of multicolored patterns and sounds generated by the performer's movement. Similarly, Vincent Vincent's demonstrations of the Mandala system carried performance art to a new level of sophistication and fantasy. The CAVE, a room with several walls of high-resolution rear-projected displays with three-dimensional audio, can offer satisfying experiences for several people at a time (Cruz-Neira et al., 1993) (Fig. 6.19).

The telepresence aspect of virtual reality breaks the physical limitations of space and allows users to act as though they are somewhere else. Practical thinkers immediately grasp the connection to remote direct manipulation, remote control, and remote vision, but the fantasists see the potential to escape current reality and to visit science-fiction worlds, cartoonlands, previous times in history, galaxies with different laws of physics, or unexplored emotional territories. Virtual worlds can be used to treat patients with fear of height by giving them an immersive experience with control over their viewpoint, while preserving their sense of physical safety (Fig. 6.20) (Hodges et al., 1995).

The direct-manipulation principles and the OAI model may be helpful to people who are designing and refining virtual environments. Users should be able to select actions rapidly by pointing or gesturing, with incremental and reversible control, and display feedback should occur immediately to convey the sense of causality. Interface objects and actions should be simple, so that users view and manipulate task-domain objects. The surgeon's instruments should be readily available or easily called up by spoken command or gesture. Similarly, an interior designer walking through a house with a client should be able to pick up a window-stretching tool or pull on a handle to try out a larger window, or to use a room-painting tool to change

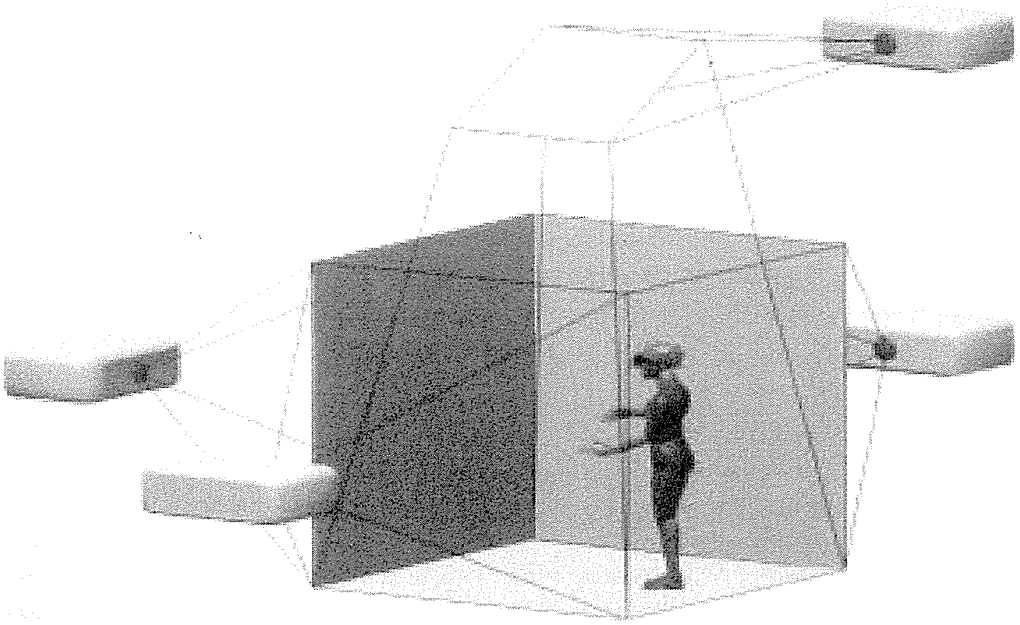


Figure 6.19

The CAVE™, a multiperson, room-sized, high-resolution, 3D video and audio environment at the University of Illinois at Chicago. The CAVE is a 10- × 10- × 9-foot theater, made up of three rear-projection screens for walls and a down-projection screen for the floor. Projectors throw full-color workstation fields (1024 × 768 stereo) onto the screens at 96 Hz. (© 1992. Image courtesy of Lewis Siegel and Kathy O'Keefe, Electronic Visualization Laboratory, University of Illinois at Chicago.)

the wall colors while leaving the windows and furniture untouched. Navigation in large virtual spaces presents further challenges, but overview maps have been demonstrated to provide useful orientation information (Darken and Sibert, 1996).

Alternatives to the immersive environment, often called *desktop* or *fishtank* virtual environments (both references are to "looking at" standard displays), are becoming more common and more accepted. The long-standing active work on three-dimensional graphics has led to user interfaces that support user-controlled exploration of real places, scientific visualizations, or fantasy worlds. Many applications run on high-performance workstations capable of rapid rendering, but some are appealing even over the web using the popular Virtual Reality Modeling Language (VRML) (Goralski, 1996).

Graphics researchers have been perfecting image display to simulate lighting effects, textured surfaces, reflections, and shadows. Data structures and algorithms for zooming in or panning across an object or room rapidly and smoothly are becoming practical on common computers. In an innova-

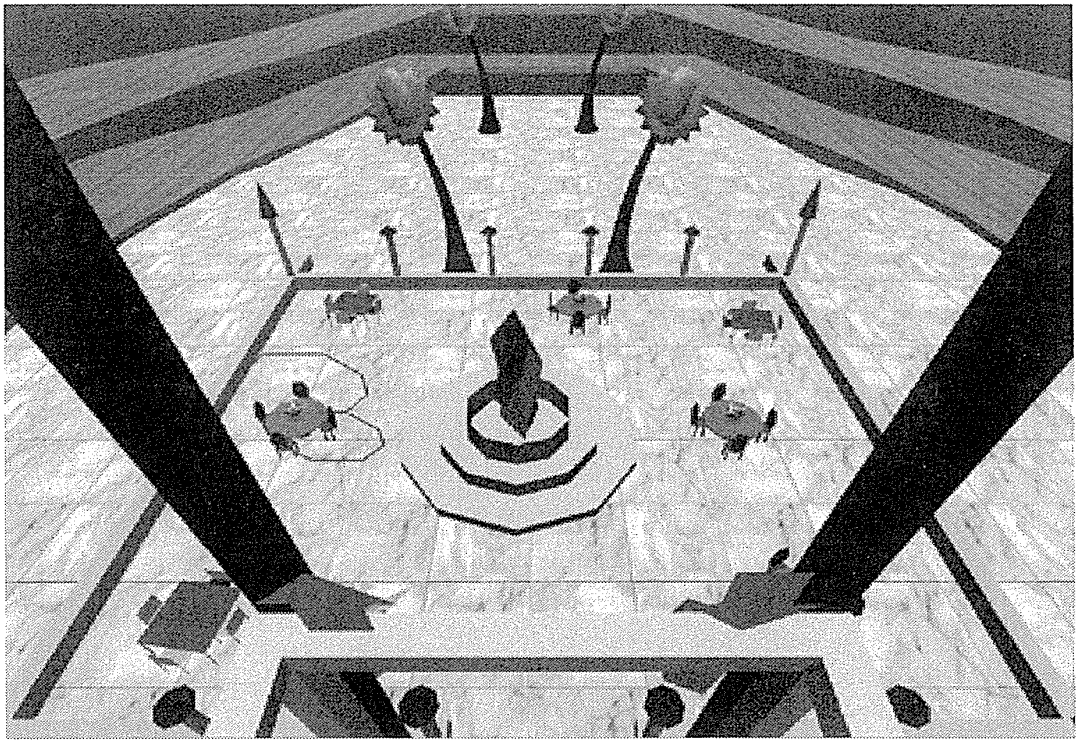


Figure 6.20

Virtual-reality therapy for users who have acrophobia. These users can accommodate to heights by going up in this virtual elevator with a guard rail located at waist level. The controls for the elevator are located on the guard rail: a green up arrow, a green down arrow, and a red stop square. (Hodges et al., 1995.) (Used with permission of Larry F. Hodges, Rob Kooper, and Tom Meyer, Georgia Tech, Atlanta, GA.)

tion called "augmented reality," users see the real world with an overlay of additional information; for example, while users are looking at the walls of a building, their semitransparent eyeglasses show where the electrical wires or plumbing are located. Augmented reality could show users where and how to repair electrical equipment or automobile engines (Feiner et al., 1993).

Another variant, called *situational awareness*, uses a palmtop computer with a location sensor to control the display. As the user moves the palmtop around a map, museum, or a piece of machinery, the display shows information about the city neighborhoods, the paintings, or the history of repairs (Fitzmaurice, 1993). Shopping carts with displays that advertise products as you walk down the supermarket aisle have already been installed.

Successful virtual environments will depend on smooth integration of multiple technologies:

- *Visual display* The normal-size (12 to 15 inches diagonally) computer display at a normal viewing distance (70 centimeters) subtends an angle of about 5 degrees; large-screen (15- to 22-inch) displays can cover a 20- to 30-degree field, and the head-mounted displays cover 100 degrees horizontally and 60 degrees vertically. The head-mounted displays block other images, so the effect is more dramatic, and head motion produces new images, so the users can get the impression of 360 degree coverage. Flight simulators also block extraneous images, but they do so without forcing the users to wear sometimes-cumbersome head-mounted displays. Another approach is a boom-mounted display that senses the users' positions without requiring that they wear heavy goggles (Fig. 6.21).

  As hardware technology improves, it will be possible to provide more rapid and higher-resolution images. Most researchers agree that the displays must approach real time (probably under 100 millisecond delay) in presenting the images to the users. Low-resolution displays are acceptable while users or the objects are moving, but when users stop to stare, higher resolution is necessary to preserve the sense of "being in." Improved hardware and algorithms are needed to display rough shapes rapidly and then to fill in the details when the motion stops. A further requirement is that motion be smooth; both incremen-

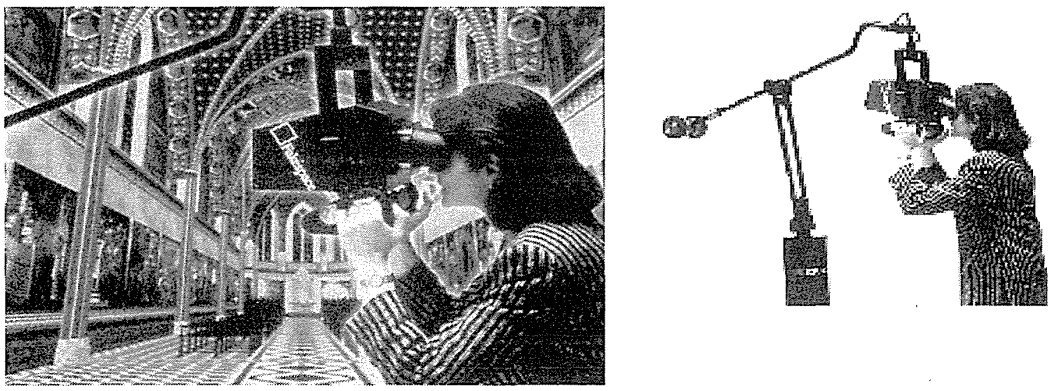


Figure 6.21

A full-color head-coupled stereoscopic display. The Fakespace BOOM3C (Binocular Omni-Orientation Monitor) provides high-quality visual displays and tracking integrated with a counterbalanced articulated arm for full six-degree of freedom motion (x, y, z, roll, pitch, yaw). Pictured here is a computer model of the Basilica of St. Francis of Assisi, complete with fourteenth century frescoes by Giotto. (Composite photo of BOOM3C® courtesy of Fakespace, Inc. (241 Polaris Avenue, Mountain View, CA 94043) and Infobyte.)

tal changes and continuous display of the objects of interest are required (Hendrix and Barfield, 1996).

- *Head-position sensing* Head-mounted displays can provide differing views depending on head position. Look to the right, and you see a forest; look to the left, and the forest gives way to a city. The Polhemus tracker requires mounting on the user's head, but other devices embedded in a hat or eyeglasses are possible. Video recognition of head position is possible. Sensor precision should be high (within 1 degree) and rapid (within 100 milliseconds). Eye tracking to recognize the focus of attention might be useful, but it is difficult to accomplish while the user is moving and is wearing a head-mounted display.
- *Hand-position sensing* The DataGlove is a highly innovative invention; it surely will be refined and improved beyond its current low resolution. Bryson (1996) complains that "the problems with glove devices include inaccuracies in measurement and lack of standard gestural vocabulary." It may turn out that accurate measurement of finger position is required only for one or two fingers or for only one or two joints. Hand orientation is provided by a Polhemus tracker mounted on the glove or wrist. Sensors for other body parts such as knees, arms, or legs may yet find uses. The potential for sensors and tactile feedback on more erotic body parts has been referred to by more than one journalist.
- *Force feedback* Hand-operated remote-control devices for performing experiments in chemistry laboratories or for handling nuclear materials provide force feedback that gives users a good sense of when they grasp an object or bump into one. Force feedback to car drivers and pilots is carefully configured to provide realistic and useful tactile information. Simulated feedback from software was successful in speeding docking tasks with complex molecules (Brooks, 1988). It might be helpful for surgeons to receive force feedback as they practice difficult operations. A palmtop display mounted on a boom was shown to produce faster and more accurate performance on a remote manipulation task when haptic (touch and force feedback) feedback was added (Noma et al., 1996). Remote handshaking as part of a video conference has been suggested, but it is not clear that the experience could be as satisfying as the real thing.
- *Sound input and output* Sound output adds realism to bouncing balls, beating hearts, or dropping vases, as videogame designers found out long ago. Making convincing sounds at the correct moment with full three-dimensional effect is possible, but it too is hard work. The digital sound hardware is adequate, but the software tools are still inadequate. Music output from virtual instruments is promising; early work simulates existing instruments such as a violin, but novel instruments have emerged. Speech recognition may complement hand gestures in some applications.

- *Other sensations* The tilting and vibration of flight simulators might provide an inspiration for some designers. Could a tilting and vibrating virtual roller coaster become popular if users could travel at 60, 600, or 6000 miles per hour and crash through mountains or go into orbit? Other effects such as a throbbing disco sound and strobe lights could also amplify some virtual experiences. Why not include real gusts of air, made hot or cold to convey the virtual weather? Finally, the power of smells to evoke strong reactions has been understood by writers from Proust to Gibson. Olfactory computing has been discussed, but appropriate and practical applications have yet to be found.
- *Cooperative and competitive virtual reality* Computer-supported cooperative work (see Chapter 14) is a lively research area, as are cooperative virtual environments, or as one developer called it, "virtuality built for two." Two people at remote sites work together, seeing each other's actions and sharing the experience. Competitive games such as virtual racquetball have been built for two players. Software for training Army tank crews took on a much more compelling atmosphere when the designs shifted from playing against the computer to shooting at other tank crews and worrying about their attacks. The realistic sounds created such a sense of engagement that crews experienced elevated heart rates, more rapid breathing, and increased perspiration. Presumably, virtual environments could also bring relaxation and pleasant encounters with other people.

6.9 Practitioner's Summary

Among interactive systems that provide equivalent functionality and reliability, some systems emerge to dominate the competition. Often, the most appealing systems have an enjoyable user interface that offers a natural representation of the task objects and actions—hence the term *direct manipulation* (Box 6.1). These systems are easy to learn, to use, and to retain over time. Novices can acquire a simple subset of the commands, and then progress to more elaborate operations. Actions are rapid, incremental, and reversible, and can be performed with physical actions instead of complex syntactic forms. The results of operations are visible immediately, and error messages are needed less often.

Just because direct-manipulation principles have been used in a system does not ensure that system's success. A poor design, slow implementation, or inadequate functionality can undermine acceptance. For some applications, menu selection, form fillin, or command languages may be more appropriate. However, the potential for direct-manipulation programming, remote direct manipulation, and virtual reality and its variants is great. Many new products will certainly emerge. Iterative design (see Chapter 3) is espe-

Box 6.1

Definition, benefits, and drawbacks of direct manipulation

Definition

- Visual representation (metaphor) of the "world of action"
  - Objects and Actions are shown
  - Analogical reasoning is tapped
- Rapid, incremental, and reversible actions
- Replacement of typing with pointing and selecting
- Immediate visibility of results of actions

Benefits over commands

- Control–display compatibility
- Less syntax reduces error rates
- Errors are more preventable
- Faster learning and higher retention
- Encourages exploration

Concerns

- Increased system resources, possibly
- Some actions may be cumbersome
- Macro techniques are often weak
- History and other tracing may be difficult
- Visually impaired users may have more difficulty

cially important in testing direct-manipulation systems, because the novelty of this approach may lead to unexpected problems for designers and users.

6.10 Researcher's Agenda

We need research to refine our understanding of the contribution of each feature of direct manipulation: analogical representation, incremental operation, reversibility, physical action instead of syntax, immediate visibility of results, and graphic form. Reversibility is easily accomplished by a generic UNDO command, but designing natural inverses for each action may be more attractive. Complex actions are well-represented with direct manipulation, but level-structured design strategies for graceful evolution from

novice to expert usage would be a major contribution. For expert users, direct-manipulation programming is still an opportunity, but good methods of history keeping and editing of action sequences are needed. Software tools to create direct-manipulation environments are sorely needed to encourage exploratory development.

Beyond the desktops, and laptops, there is the allure of telepresence, virtual environments, augmented realities, and situationally aware devices. The playful aspects will certainly be pursued, but the challenge is to find the practical designs for being in and looking at three-dimensional worlds. Novel devices for walking through museums or supermarkets and teleoperation for repair seem good candidates for entrepreneurs.

World Wide Web Resources WWW

Some creative direct manipulation services and tools are linked to, but the majority of links cover direct manipulation programming, teleoperation, and virtual environments. The web-based Virtual Reality Modeling Language enables creation of three-dimensional environments on web pages and there are numerous visually appealing websites.

http://www.aw.com/DTUI

References

Arnheim, Rudolf, *Visual Thinking*, University of California Press, Berkeley, CA (1972).

Benbasat, Izak and Todd, P., An experimental investigation of interface design alternatives: Icon versus text and direct manipulation versus menus, *International Journal of Man–Machine Studies*, 38, 3 (1993), 369–402.

Brooks, Frederick, Grasping reality through illusion: Interactive graphics serving science, *Proc. CHI '88 Conference—Human Factors in Computing Systems*, ACM, New York (1988), 1–11.

Bruner, James, *Toward a Theory of Instruction*, Harvard University Press, Cambridge, MA (1966).

Bryson, Steve, Virtual reality in scientific visualization, *Communications of the ACM*, 39, 5 (May 1996), 62–71.

Carroll, John M. and Thomas, John C., Metaphor and the cognitive representation of computing systems, *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-12, 2 (March–April 1982), 107–116.

Carroll, J. M., Thomas, J. C., and Malhotra, A., Presentation and representation in design problem-solving, *British Journal of Psychology*, 71, (1980), 143–153.

Copeland, Richard W., *How Children Learn Mathematics* (Third Edition), MacMillan, New York (1979).

Cruz-Neira, C., Sandin, D. J., and DeFanti, T., Surround-screen projection-based virtual reality: The design and implementation of the CAVE, *Proc. SIGGRAPH '93 Conference*, ACM, New York (1993), 135–142.

Cypher, Allen, EAGER: Programming repetitive tasks by example, *Proc. CHI '91 Conference—Human Factors in Computing Systems*, ACM, New York (1991), 33–39.

Darken, Rudolph, P. and Sibert, John L., Navigating large virtual spaces, *International Journal of Human–Computer Interaction*, 8, 1 (1996), 49–71.

Feiner, Steven, MacIntyre, Blair, and Seligmann, Doree, Knowledge-based augmented reality, *Communications of the ACM*, 36, 7 (1993), 52–62.

Fitzmaurice, George, Situated information spaces and spatially aware palmtop computers, *Communications of the ACM*, 36, 7 (1993), 39–49.

Frohlich, David M., The history and future of direct manipulation, *Behaviour and Information Technology*, 12, 6 (1993), 315–329.

Goralski, Walter, *VRML: Exploring Virtual Worlds on the Internet*, Prentice Hall, Englewood Cliffs, NJ (1996).

Green, T. R. G. and Petre, M., Usability analysis of visual programming environments: A "cognitive dimensions" framework, *Journal of Visual Languages and Computing*, 7, (1996), 131–174.

Heckel, Paul, *The Elements of Friendly Software Design: The New Edition*, SYBEX, San Francisco (1991).

Hendrix, C., and Barfield, W., Presence within virtual environments as a function of visual display parameters, *Presence: Teleoperators and Virtual Environments*, 5, 3 (1996), 274–289.

Herot, Christopher F., Spatial management of data, *ACM Transactions on Database Systems*, 5, 4, (December 1980), 493–513.

Herot, Christopher, Graphical user interfaces. In Vassiliou, Yannis (Editor), *Human Factors and Interactive Computer Systems*, Ablex, Norwood, NJ (1984), 83–104.

Hinckley, Ken, Pausch, Randy, Goble, John C., and Kassell, Neal F., Passive real-world props for neurosurgical visualization, *Proc. CHI '94 Conference—Human Factors in Computing Systems*, ACM, New York (1994), 452–458.

Hodges, L.F., Rothbaum, B.O., Kooper, R., Opdyke, D., Meyer, T., North, M., de Graff, J.J., and Williford, J., Virtual environments for treating the fear of heights, *IEEE Computer*, 28, 7 (1995), 27–34.

Hutchins, Edwin L., Hollan, James D., and Norman, Don A., Direct manipulation interfaces. In Norman, Don A. and Draper, Stephen W. (Editors), *User Centered System Design: New Perspectives on Human–Computer Interaction*, Lawrence Erlbaum Associates, Hillsdale, NJ (1986), 87–124.

Iseki, Osamu and Shneiderman, Ben, Applying direct manipulation concepts: Direct Manipulation Disk Operating System (DMDOS), *Software Engineering Notes*, 11, 2, (March 1986), 22–26.

Krueger, Myron, *Artificial Reality II*, Addison-Wesley, Reading, MA (1991).

Laurel, Brenda, *Computers as Theatre*, Addison-Wesley, Reading, MA (1991).

MacDonald, Lindsay and Vince, John (Editors), *Interacting with Virtual Environments*, John Wiley and Sons, New York (1994).

McKim, Robert H., *Experiences in Visual Thinking* (Second Edition), Brooks/Cole, Monterey, CA (1980).

Malone, Thomas W., What makes computer games fun? *BYTE*, 6, 12 (December 1981), 258–277.

Marcus, Aaron, *Graphic Design for Electronic Documents and User Interfaces*, ACM Press, New York (1992).

Margono, Sepeedeh and Shneiderman, Ben, A study of file manipulation by novices using commands versus direct manipulation, *Twenty-sixth Annual Technical Symposium*, ACM, Washington, D.C. (June 1987), 154–159.

Maulsby, David L. and Witten, Ian H., Inducing programs in a direct-manipulation environment, *Proc. CHI '89 Conference—Human Factors in Computing Systems*, ACM, New York (1989), 57–62.

Montessori, Maria, *The Montessori Method*, Schocken, New York (1964).

Morgan, K., Morris, R. L., and Gibbs, S., When does a mouse become a rat? or . . . Comparing performance and preferences in direct manipulation and command line environment, *The Computer Journal*, 34, 3 (1991), 265–271.

Mullet, Kevin and Sano, Darrell, *Designing Visual Interfaces: Communication Oriented Techniques*, Sunsoft Press, Englewood Cliffs, NJ (1995).

Myers, Brad A., Demonstrational interfaces: A step beyond direct manipulation, *IEEE Computer*, 25, 8 (August 1992), 61–73.

Nelson, Ted, Interactive systems and the design of virtuality, *Creative Computing*, 6, 11, (November 1980), 56 ff., and G, 12 (December 1980), 94 ff.

Noma, Haruo, Miyasato, Tsutomu, and Kishino, Fumio, A palmtop display for dexterous manipulation with haptic sensation, *Proc. CHI '96 Conference—Human Factors in Computing Systems*, ACM, New York (1996), 126–133.

Norman, Donald A., *The Psychology of Everyday Things*, Basic Books, New York (1988).

Norman, Kent, *The Psychology of Menu Selection: Designing Cognitive Control at the Human/Computer Interface*, Ablex, Norwood, NJ (1991).

Papert, Seymour, *Mindstorms: Children, Computers, and Powerful Ideas*, Basic Books, New York (1980).

Phillips, C. H. E. and Apperley, M. D., Direct manipulation interaction tasks: A Macintosh-based analysis, *Interacting with Computers*, 3, 1 (1991), 9–26.

Plaisant, Catherine and Shneiderman, Ben, Scheduling ON–OFF home control devices: Design issues and usability evaluation of four touchscreen interfaces, *International Journal for Man–Machine Studies*, 36, (1992), 375–393.

Plaisant, C., Shneiderman, B., and Battaglia, J., Scheduling home-control devices: A case study of the transition from the research project to a product, *Human-Factors in Practice*, Computer Systems Technical Group, Human-Factors Society, Santa Monica, CA (December 1990), 7–12.

Polya, G., *How to Solve It*, Doubleday, New York, (1957).

Potter, Richard, Just in Time programming. In Cypher, Allen (Editor), *Watch What I Do: Programming by Demonstration*, MIT Press, Cambridge, MA (1993), 513–526.

Provenzo, Jr., Eugene R., *Video Kids: Making Sense of Nintendo*, Harvard University Press, Cambridge, MA (1991).

Rheingold, Howard, *Virtual Reality*, Simon and Schuster, New York (1991).

Robertson, George G., Card, Stuart K., and Mackinlay, Jock D., Information visualization using 3-D interactive animation, *Communications of the ACM*, 36, 4 (April 1993), 56–71.

Rogers, Yvonne, Icons at the interface: Their usefulness, *Interacting with Computers*, 1, 1 (1989), 105–117.

Rubin, Robert V., Golin, Eric J., and Reiss, Steven P., Thinkpad: A graphics system for programming by demonstrations, *IEEE Software*, 2, 2 (March 1985), 73–79.

Rutkowski, Chris, An introduction to the Human Applications Standard Computer Interface, Part 1: Theory and principles, *BYTE*, 7, 11 (October 1982), 291–310.

Satava, R. M. and Jones, S. B., Virtual reality and telemedicine: Exploring advanced concepts, *Telemedicine Journal*, 2, 3 (1996), 195–200.

Sheridan, T. B., *Telerobotics, Automation, and Human Supervisory Control*, The MIT Press, Cambridge, MA (1992).

Shneiderman, Ben, Direct manipulation: A step beyond programming languages, *IEEE Computer*, 16, 8, (August 1983), 57–69.

Smith, David Canfield, *Pygmalion: A Computer Program to Model and Stimulate Creative Thought*, Birkhauser Verlag, Basel, Switzerland (1977).

Smith, D. Canfield, Irby, Charles, Kimball, Ralph, Verplank, Bill, and Harslem, Eric, Designing the Star user interface, *BYTE*, 7, 4 (April 1982), 242–282.

Stuart, Rory, *The Design of Virtual Environments*, McGraw-Hill, New York (1996).

Temple, Barker, and Sloane, Inc., The benefits of the graphical user interface, *Multimedia Review* (Winter 1990), 10–17.

Thimbleby, Harold, *User Interface Design*, ACM Press, New York (1990).

Ulich, E., Rauterberg, M., Moll, T., Greutmann, T., and Strohm, O., Task orientation and user-orientated dialogue design, *International Journal of Human–Computer Interaction*, 3, 2 (1991), 117–144.

Uttal, W. R., Teleoperators, *Scientific American*, 261, 6 (December 1989), 124–129.

Vince, John, *Virtual Reality Systems*, Addison-Wesley, Reading, MA (1995).

Van de Vegte, J. M. E., Milgram, P., Kwong, R. H., Teleoperator control models: Effects of time delay and imperfect system knowledge, *IEEE Transactions on Systems, Man, and Cybernetics*, 20, 6 (November–December 1990), 1258–1272.

Verplank, William L., Graphic challenges in designing object-oriented user interfaces. In Helander, M. (Editor), *Handbook of Human–Computer Interaction*, Elsevier Science Publishers, Amsterdam, The Netherlands (1988), 365–376.

Weinstein, R., Bloom, K., Rozek, S., Telepathology: Long distance diagnosis, *American Journal of Clinical Pathology*, 91 (Suppl 1) (1989), S39–S42.

Wertheimer, M., *Productive Thinking*, Harper and Row, New York (1959).

Ziegler, J. E. and Fähnrich, K.-P., Direct manipulation. In Helander, M. (Editor), *Handbook of Human–Computer Interaction*, Elsevier Science Publishers, Amsterdam, The Netherlands (1988), 123–133.