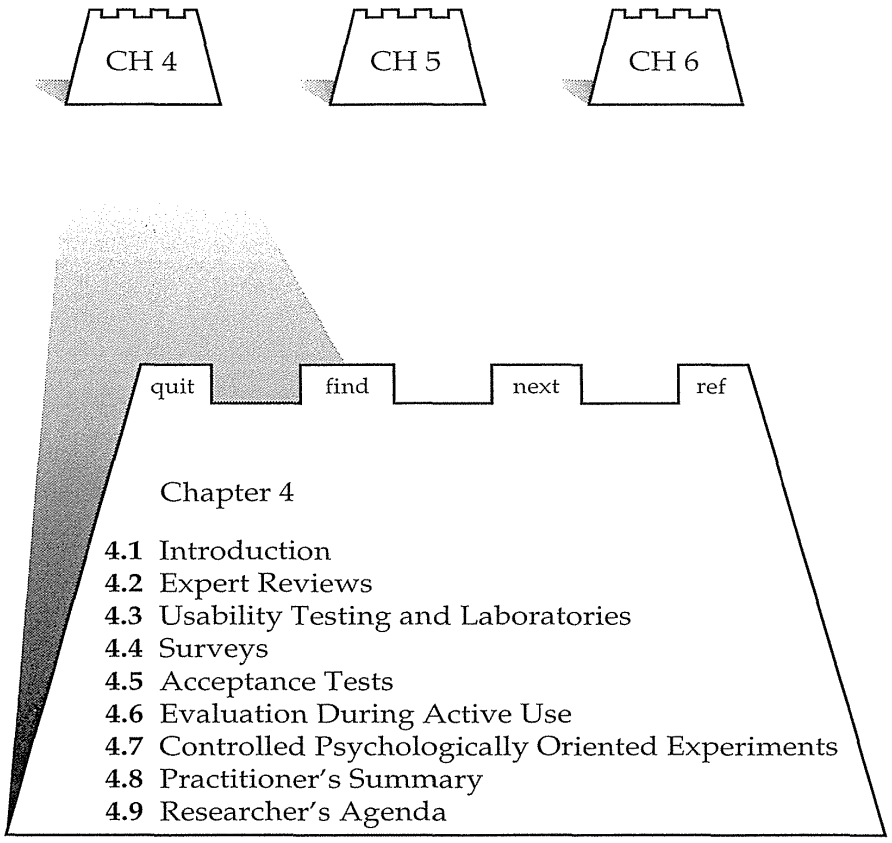
CHAPTER

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-

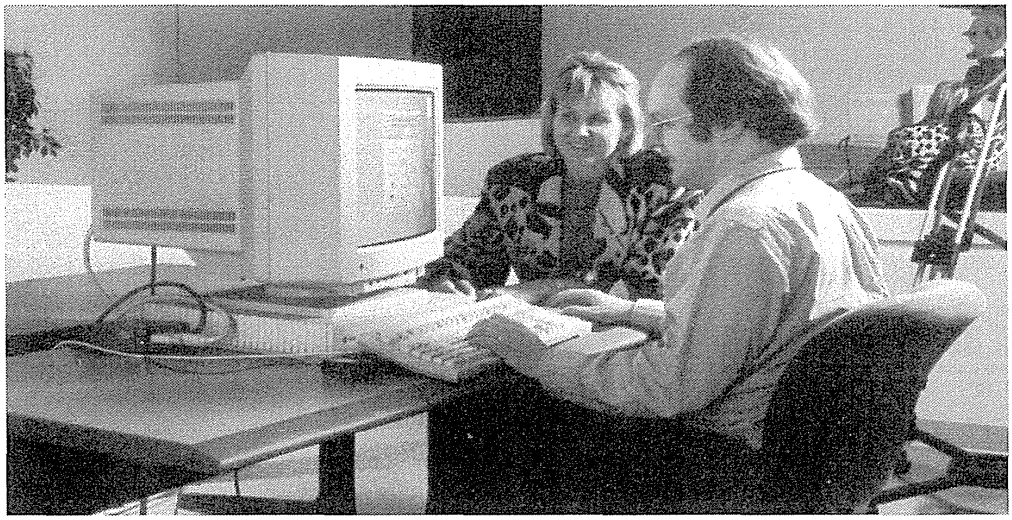


Figure 4.1

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

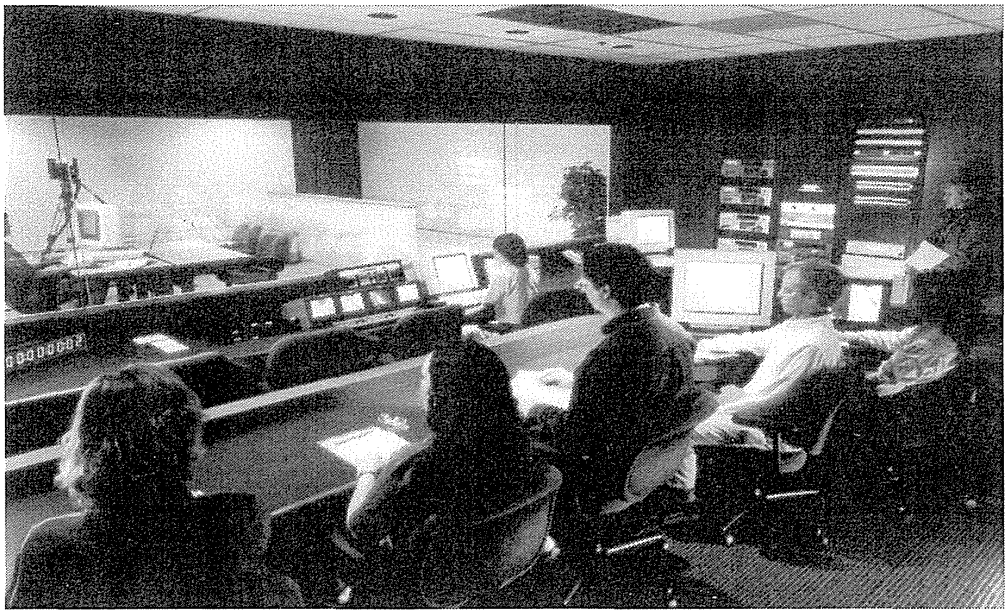


Figure 4.2

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	always	
	1 2 3 4 5 6 7 8 9		NA
5.2.1 Computer terminology is used	too frequently	appropriately	
	1 2 3 4 5 6 7 8 9		NA
5.2.2 Terminology on the screen	ambiguous	precise	
	1 2 3 4 5 6 7 8 9		NA
5.3 Messages which appear on screen	inconsistent	consistent	
	1 2 3 4 5 6 7 8 9		NA
5.3.1 Position of instructions on the screen	inconsistent	consistent	
	1 2 3 4 5 6 7 8 9		NA
5.4 Messages which appear on screen	confusing	clear	
	1 2 3 4 5 6 7 8 9		NA
5.4.1 Instructions for commands or functions	confusing	clear	
	1 2 3 4 5 6 7 8 9		NA
5.4.2 Instructions for correcting errors	confusing	clear	
	1 2 3 4 5 6 7 8 9		NA
5.5 Computer keeps you informed about what it is doing	never	always	
	1 2 3 4 5 6 7 8 9		NA
5.5.1 Animated cursors keep you informed	never	always	
	1 2 3 4 5 6 7 8 9		NA
5.5.2 Performing an operation leads to a predictable result	never	always	
	1 2 3 4 5 6 7 8 9		NA
5.5.3 Controlling amount of feedback	impossible	easy	
	1 2 3 4 5 6 7 8 9		NA
5.5.4 Length of delay between operations	unacceptable	acceptable	
	1 2 3 4 5 6 7 8 9		NA
5.6 Error messages	unhelpful	helpful	
	1 2 3 4 5 6 7 8 9		NA
5.6.1 Error messages clarify the problem	never	always	
	1 2 3 4 5 6 7 8 9		NA
5.6.2 Phrasing of error messages	unpleasant	pleasant	
	1 2 3 4 5 6 7 8 9		NA

Please write your comments about terminology and system information here:

PART 6: Learning

6.1 Learning to operate the system	difficult	easy	
	1 2 3 4 5 6 7 8 9		NA
6.1.1 Getting started	difficult	easy	
	1 2 3 4 5 6 7 8 9		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	always	
	1 2 3 4 5 6 7 8 9		NA
11.3 Determining the focus of attention during conference was	confusing	clear	
	1 2 3 4 5 6 7 8 9		NA
11.3.1 Telling who is speaking	difficult	easy	
	1 2 3 4 5 6 7 8 9		NA
11.4 Video image flow	choppy	smooth	
	1 2 3 4 5 6 7 8 9		NA
11.4.1 Focus of video image	fuzzy	clear	
	1 2 3 4 5 6 7 8 9		NA
11.5 Audio output	inaudible	audible	
	1 2 3 4 5 6 7 8 9		NA
11.5.1 Audio is in sync with video images	never	always	
	1 2 3 4 5 6 7 8 9		NA
11.6 Exchanging data	difficult	easy	
	1 2 3 4 5 6 7 8 9		NA
11.6.1 Transmitting files	difficult	easy	
	1 2 3 4 5 6 7 8 9		NA
11.6.2 Retrieving files	difficult	easy	
	1 2 3 4 5 6 7 8 9		NA
11.6.3 Using on-line chat	difficult	easy	
	1 2 3 4 5 6 7 8 9		NA
11.6.4 Using shared workspace	difficult	easy	
	1 2 3 4 5 6 7 8 9		NA

Please write your comments about teleconferencing here:

PART 12: Software Installation

12.1 Speed of installation	slow	fast	
	1 2 3 4 5 6 7 8 9		NA
12.2 Customization	difficult	easy	
	1 2 3 4 5 6 7 8 9		NA
12.2.1 Installing only the software you want	confusing	clear	
	1 2 3 4 5 6 7 8 9		NA
12.3 Informs you of its progress	never	always	
	1 2 3 4 5 6 7 8 9		NA
12.4 Gives a meaningful explanation when failures occur	never	always	
	1 2 3 4 5 6 7 8 9		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).

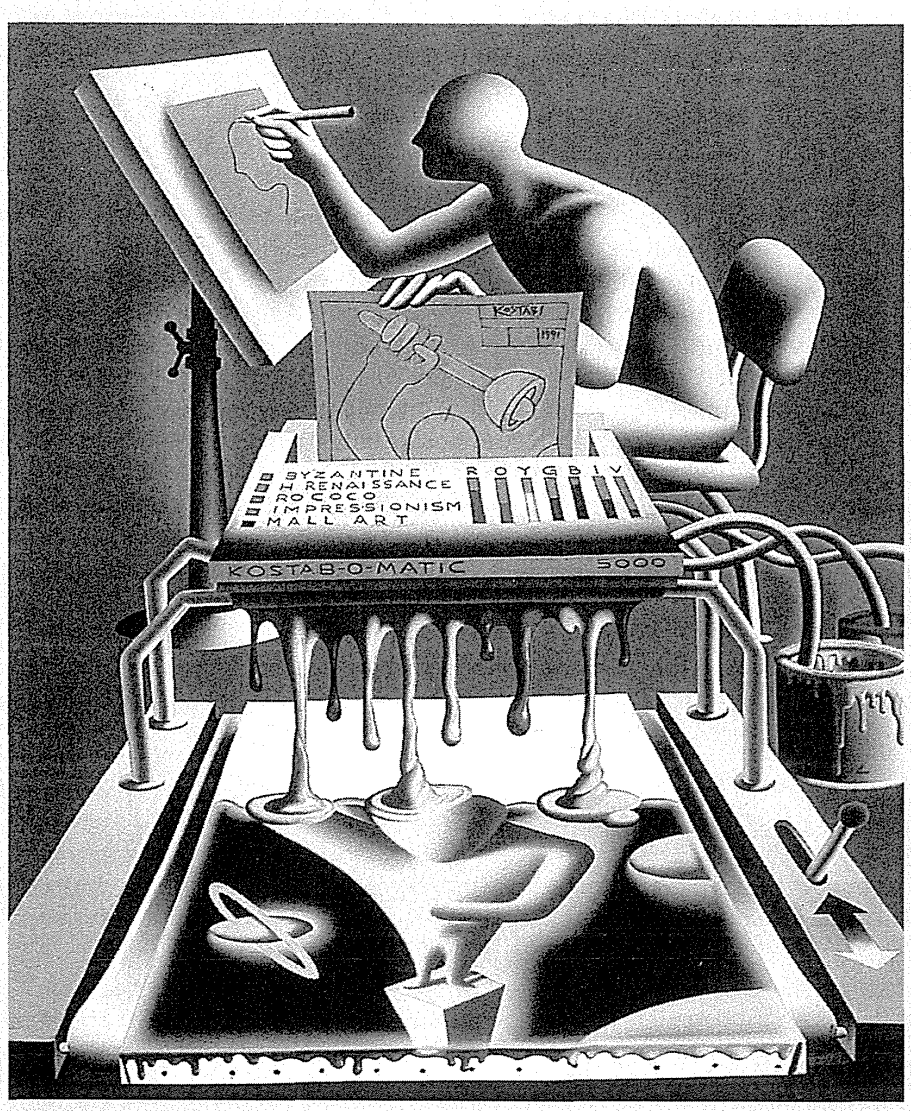
Preece, Jenny, Rogers, Yvonne, Sharp, Helen, Benyon, David, Holland, Simon, and Carey, Tom, *Human–Computer Interaction*, Addison-Wesley, Reading, MA (1994).

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

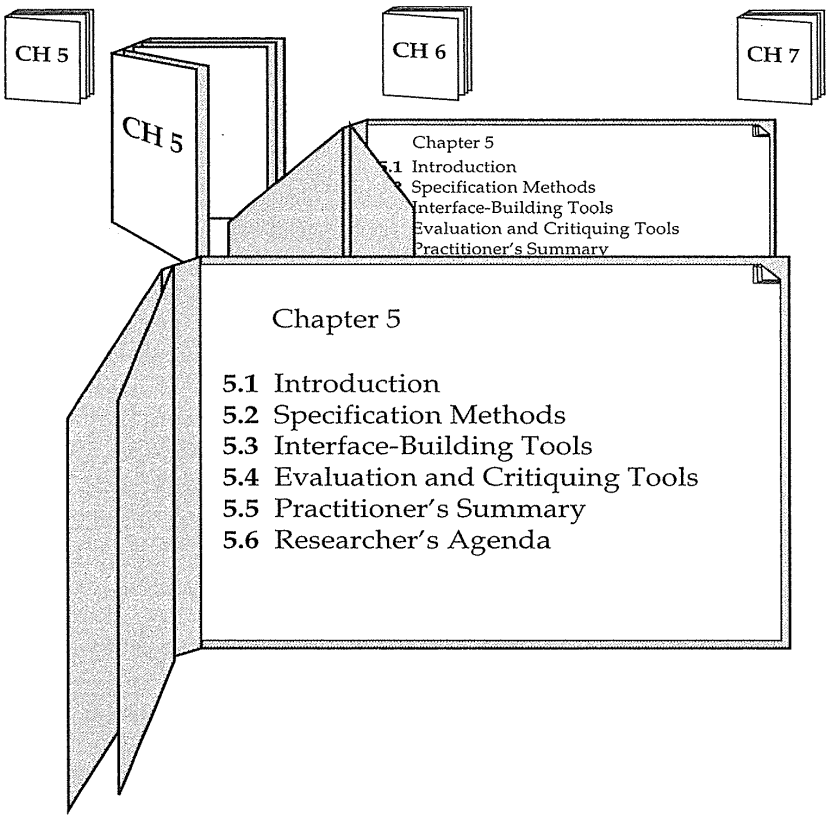
CHAPTER

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-

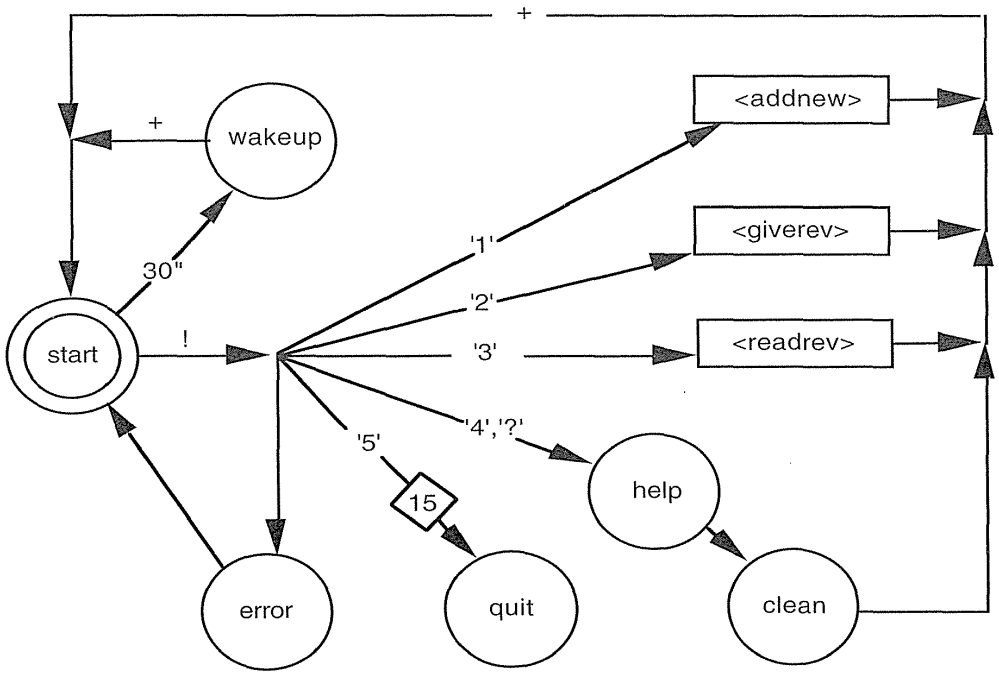


Figure 5.1

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$-1, cl,r$,cl
node wakeup
       r$,cl,rv,'Please make a choice',sv,  tomark_A
node quit
       cs,  'Thank you very much.  Please try this program again',
       nl,'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

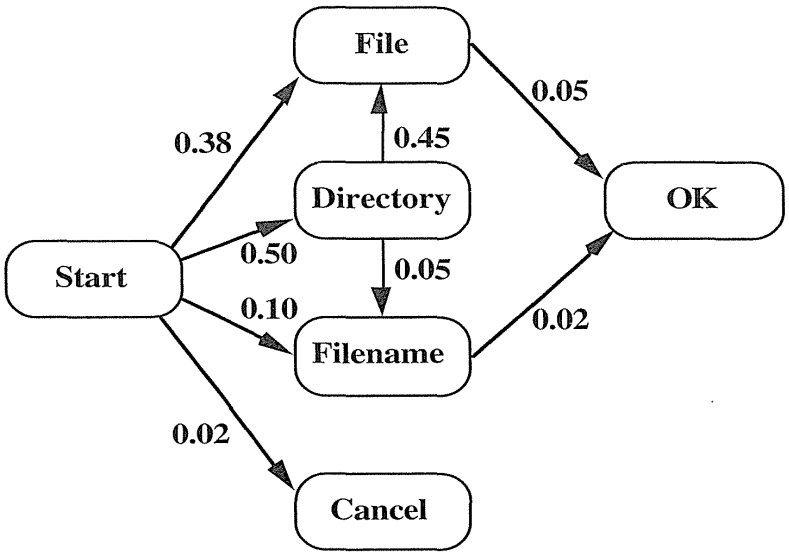


Figure 5.3

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

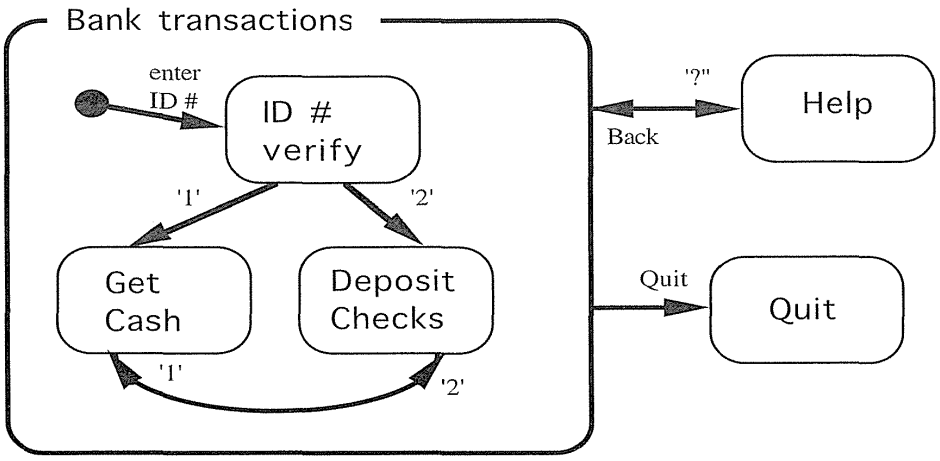


Figure 5.4

Statechart of a simplified bank transaction system showing grouping of states.

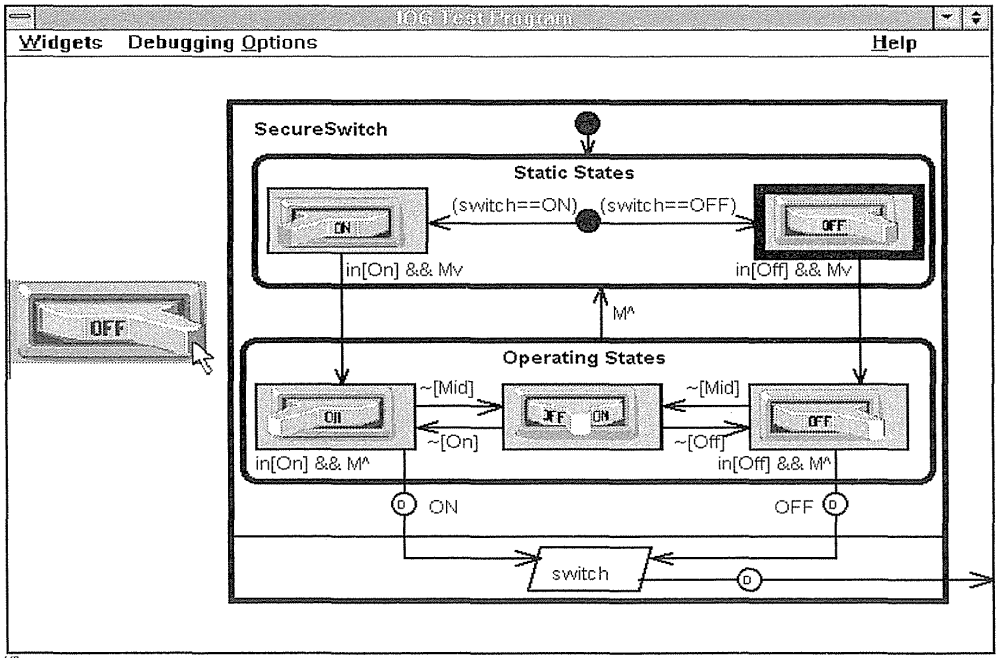


Figure 5.5

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).

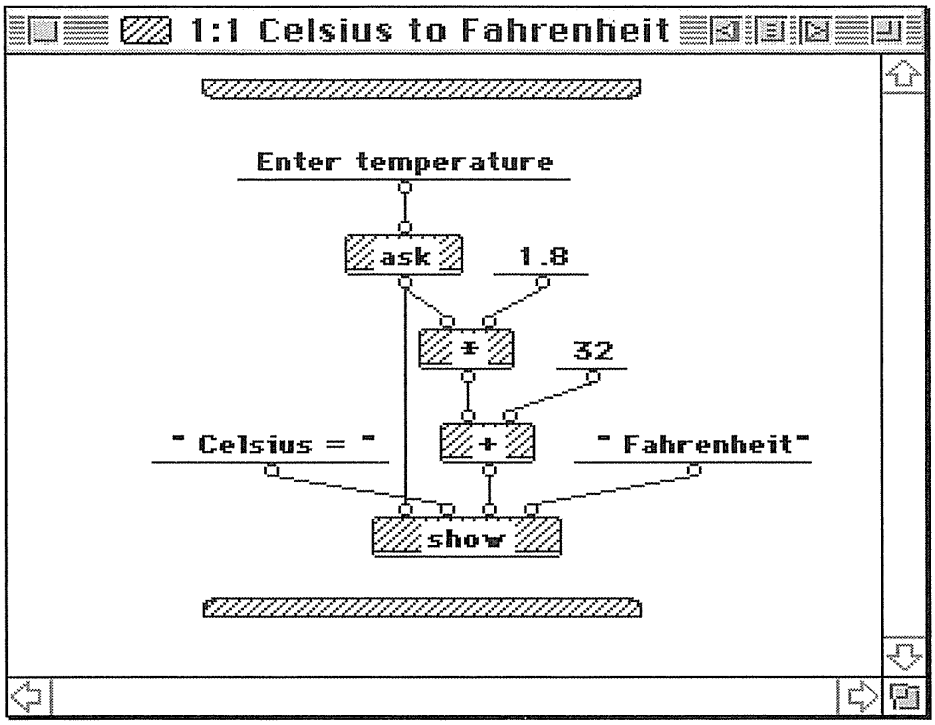


Figure 5.6

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-

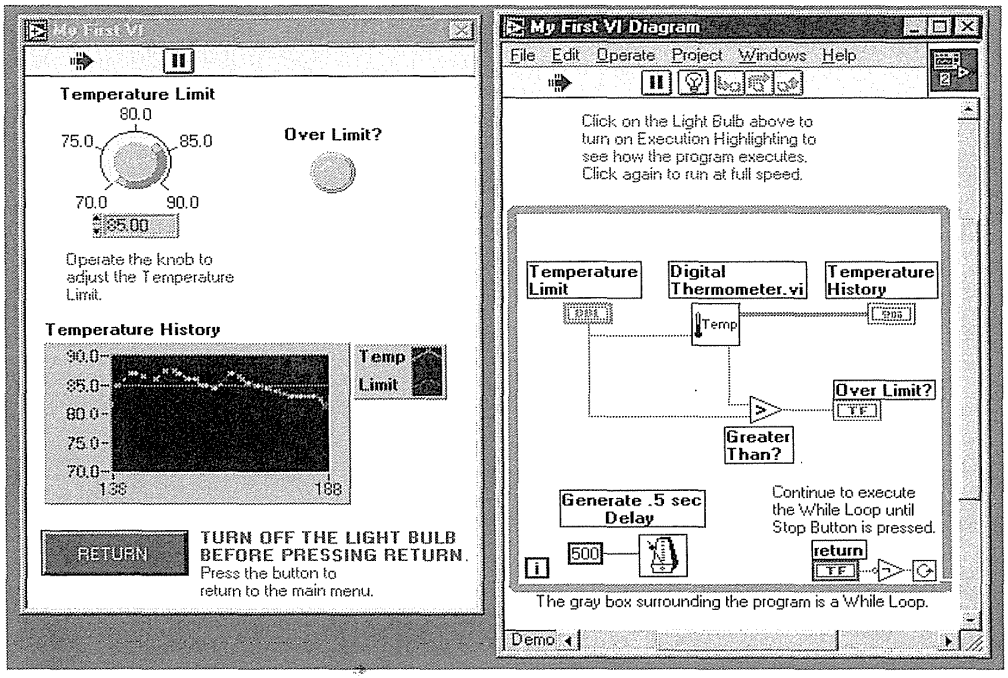


Figure 5.7

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):

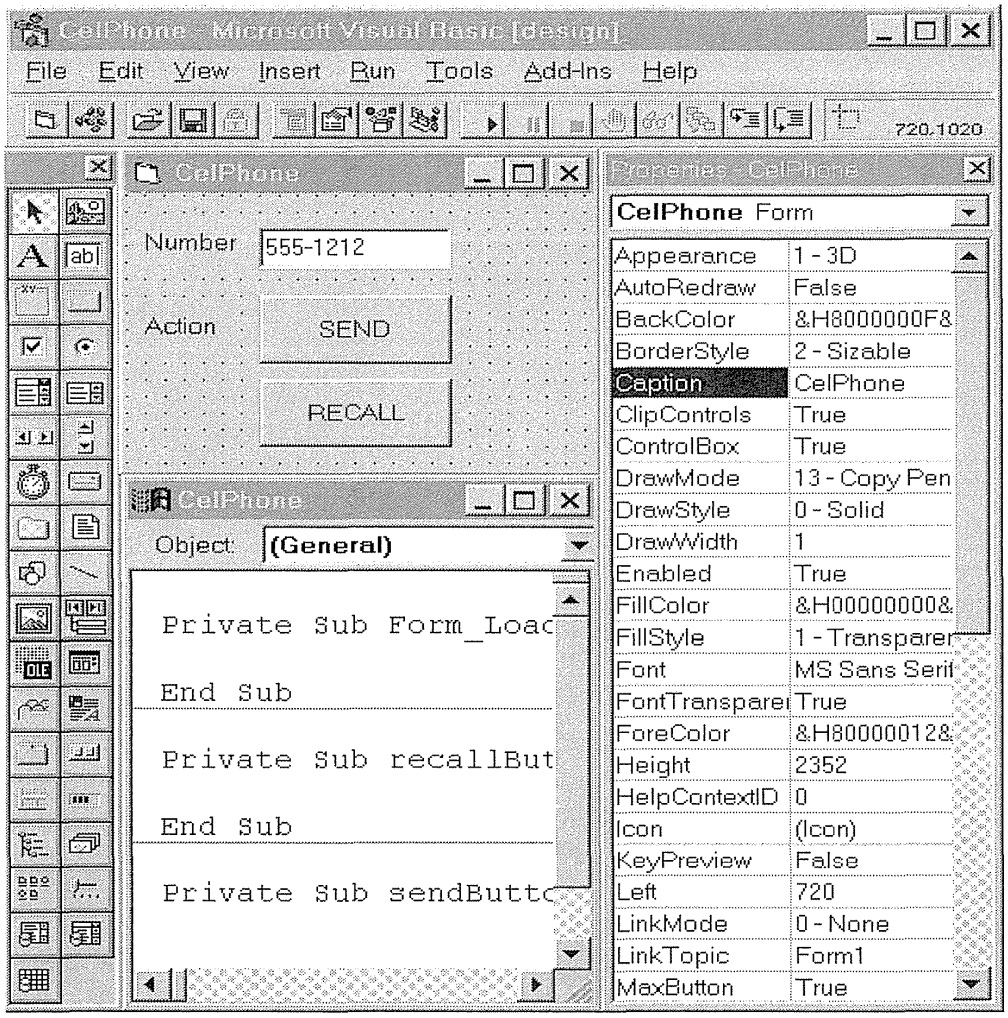


Figure 5.8

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.)

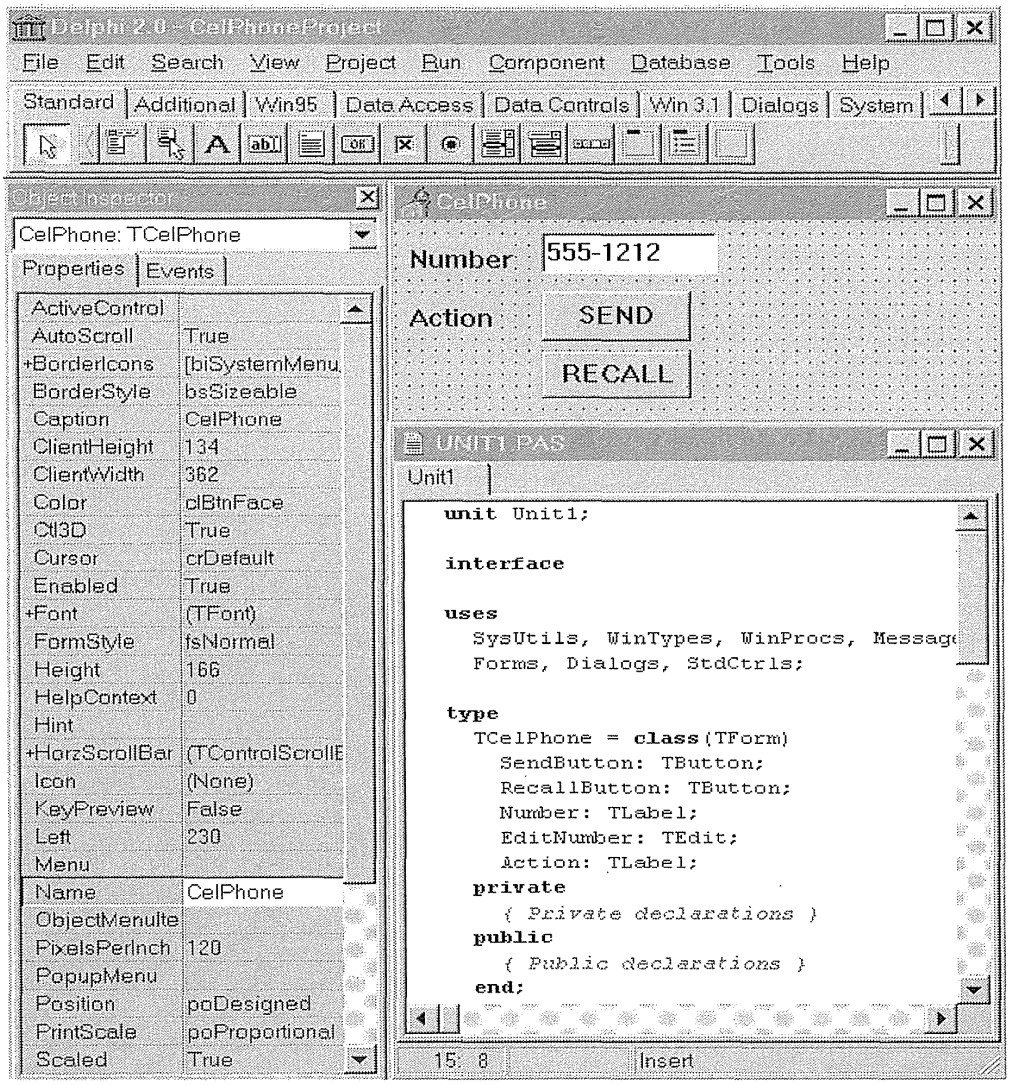


Figure 5.9

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)

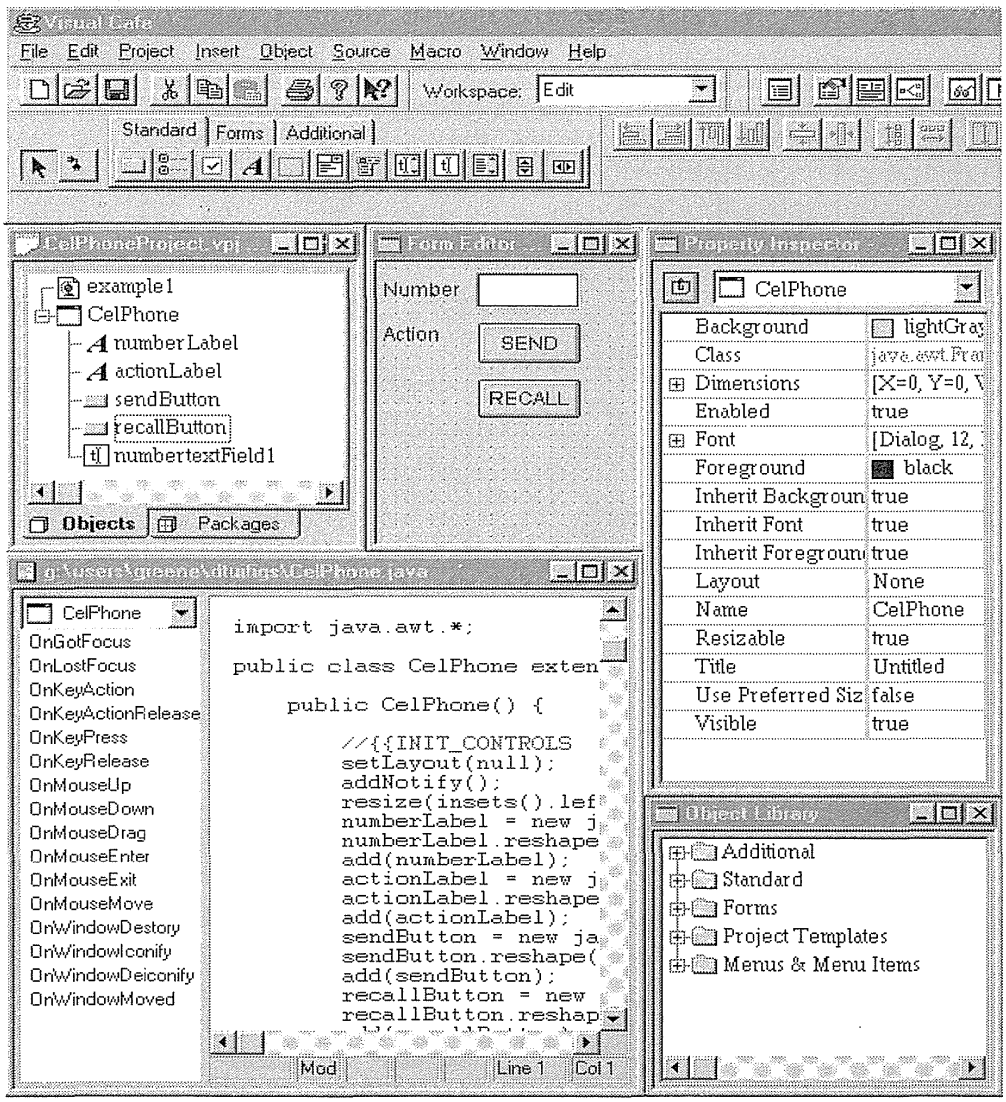


Figure 5.10

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.

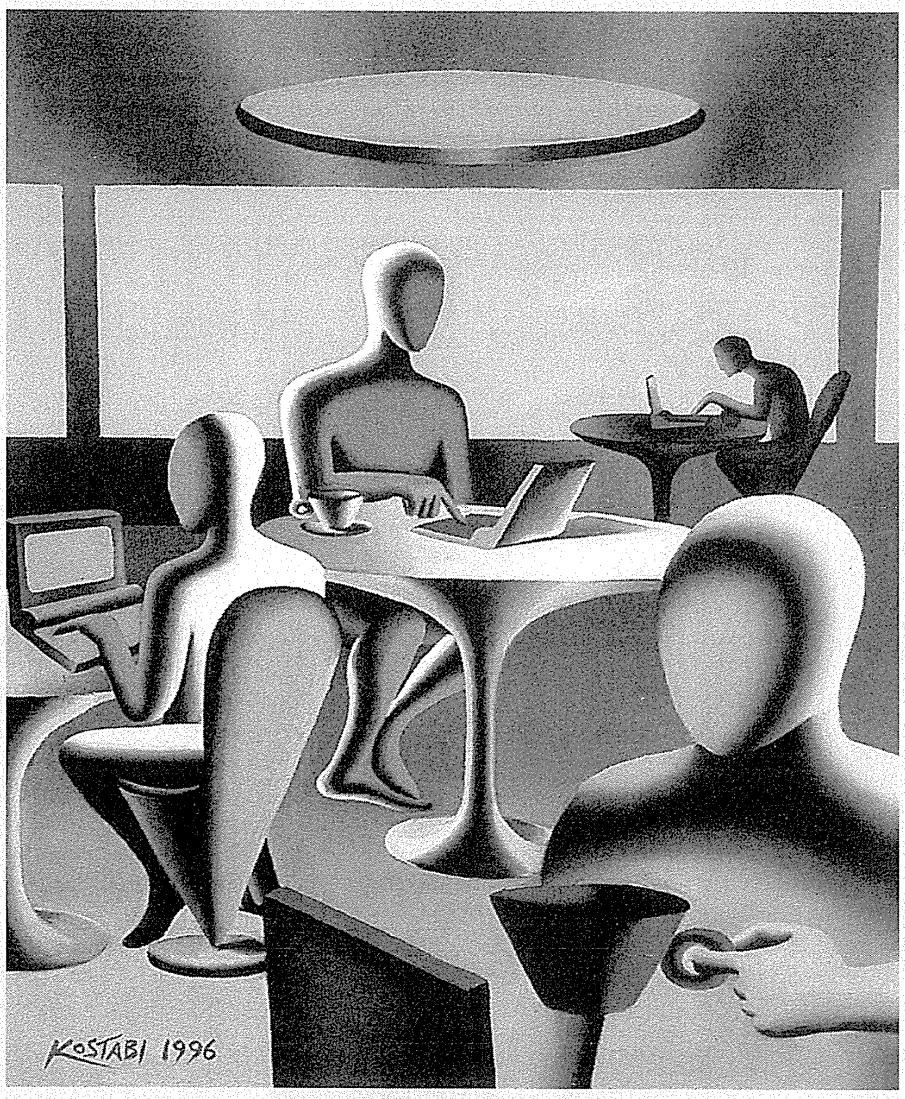
Sears, Andrew, Layout appropriateness: Guiding user interface design with simple task descriptions, *IEEE Transactions on Software Engineering*, 19, 7 (1993), 707–719.

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

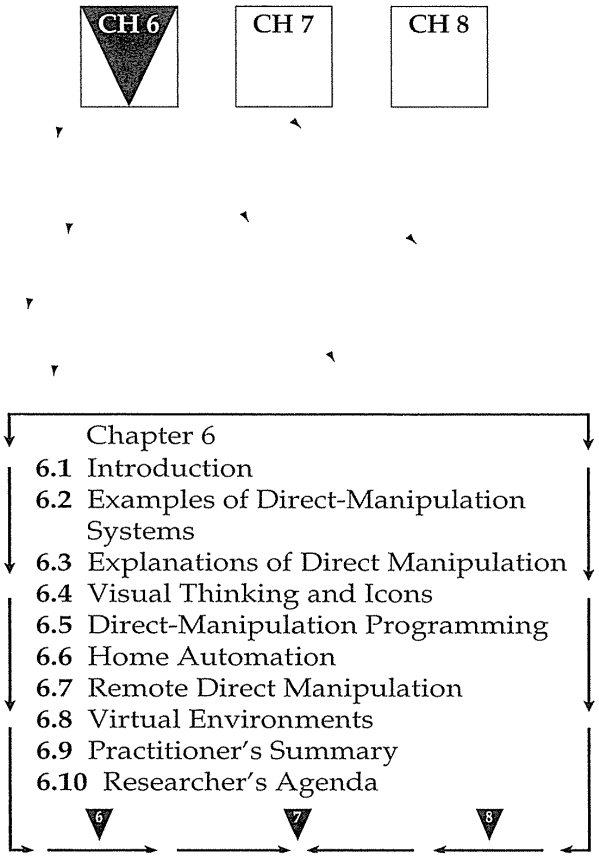
CHAPTER

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—

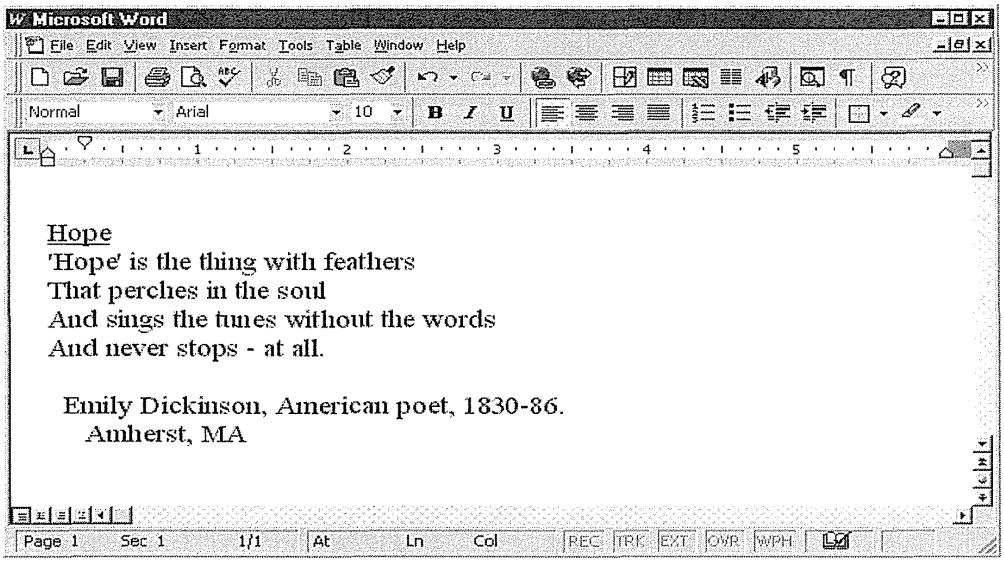


Figure 6.1

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.

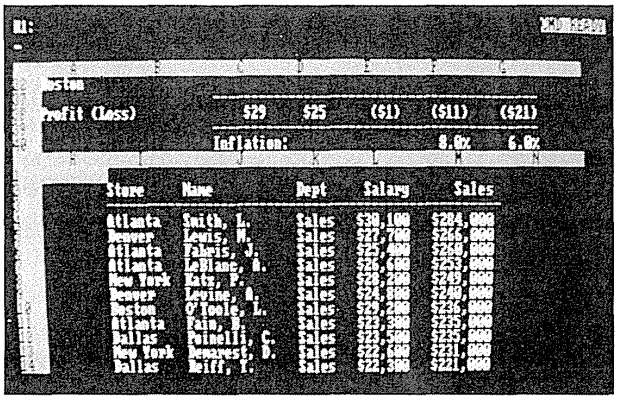
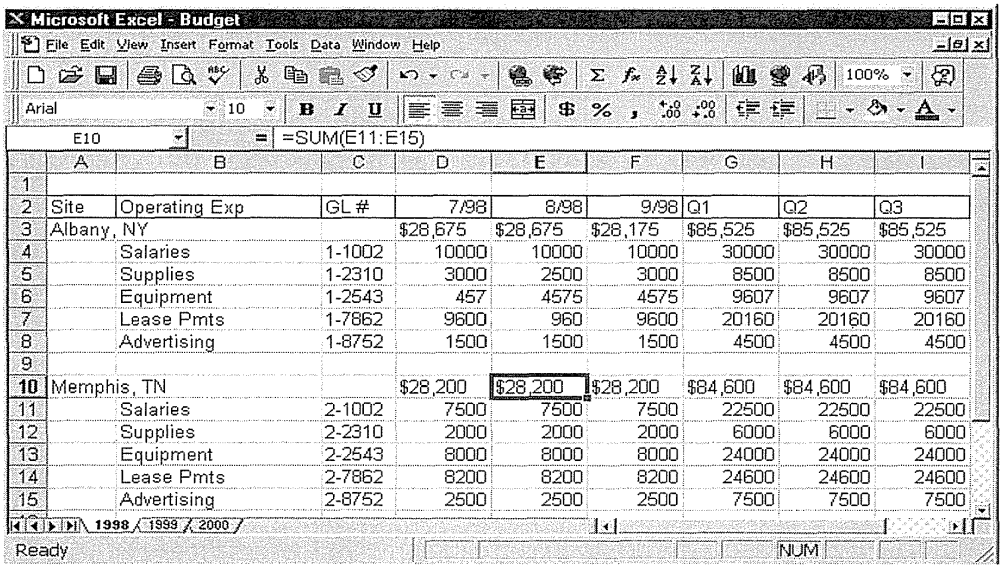


Figure 6.2

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

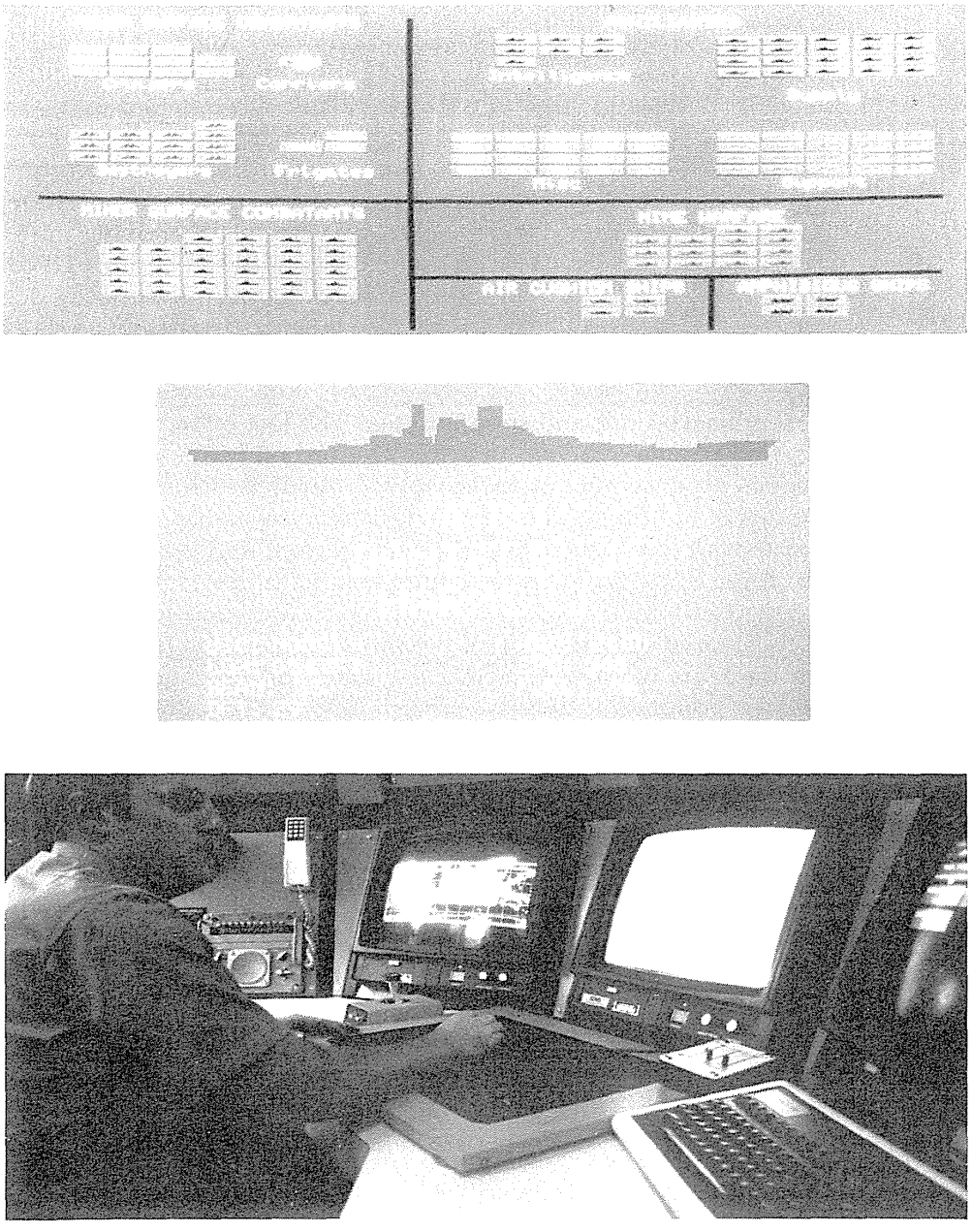


Figure 6.4

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.)

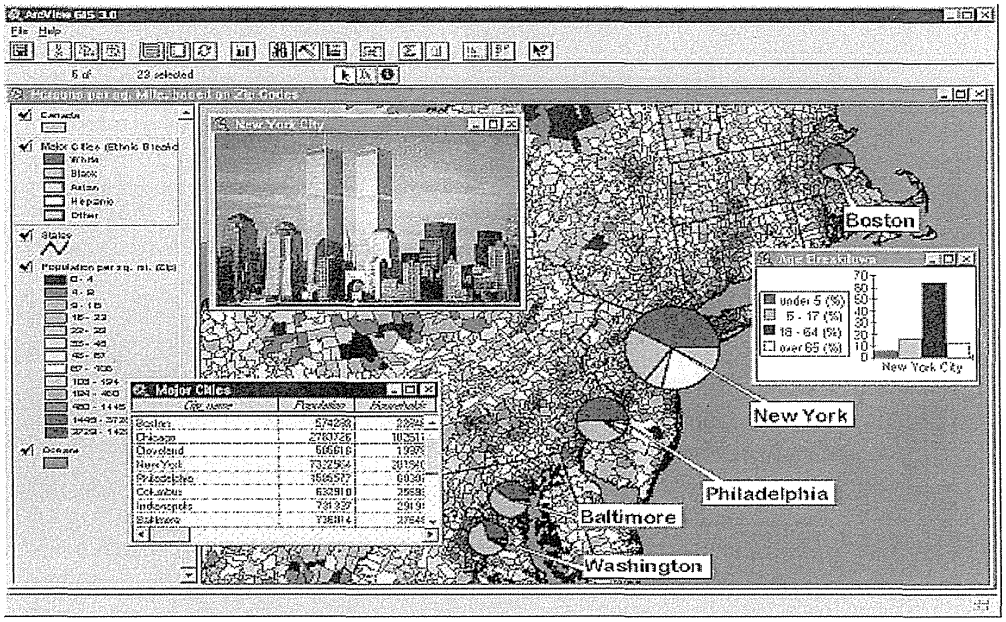


Figure 6.5

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

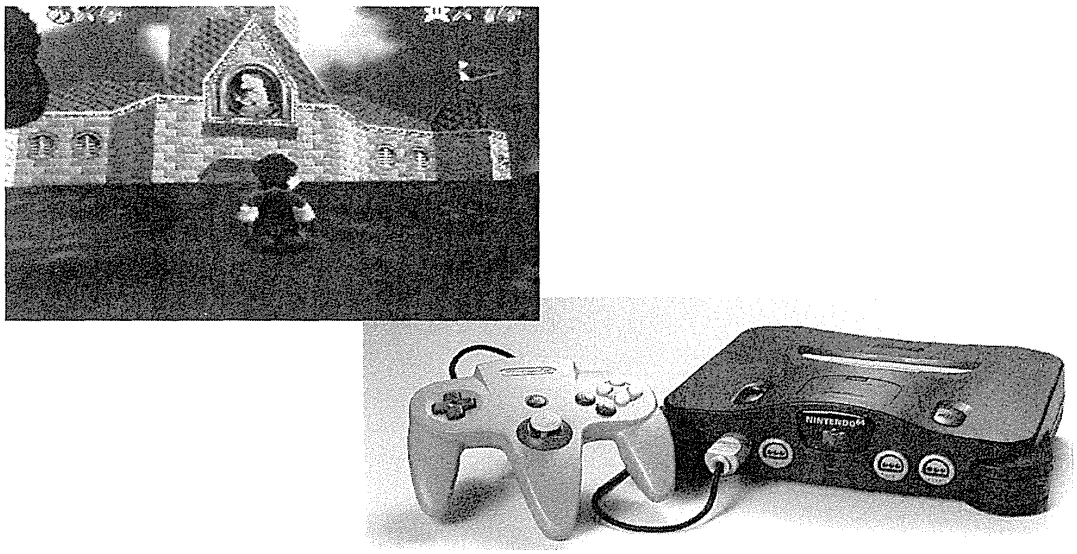


Figure 6.6

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

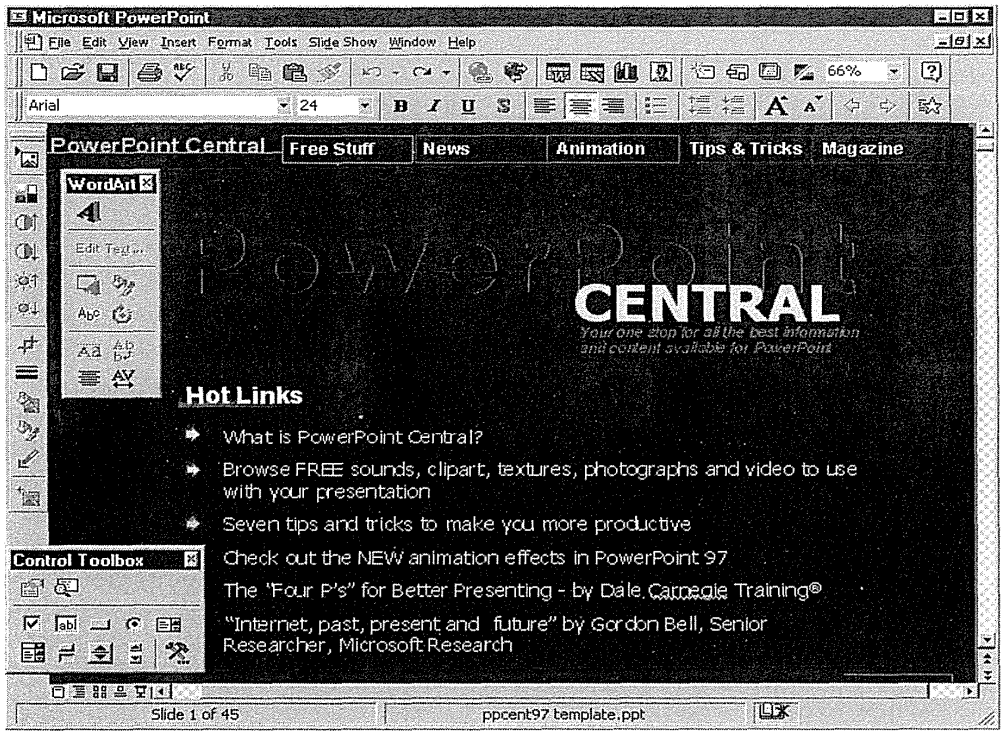


Figure 6.7

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).

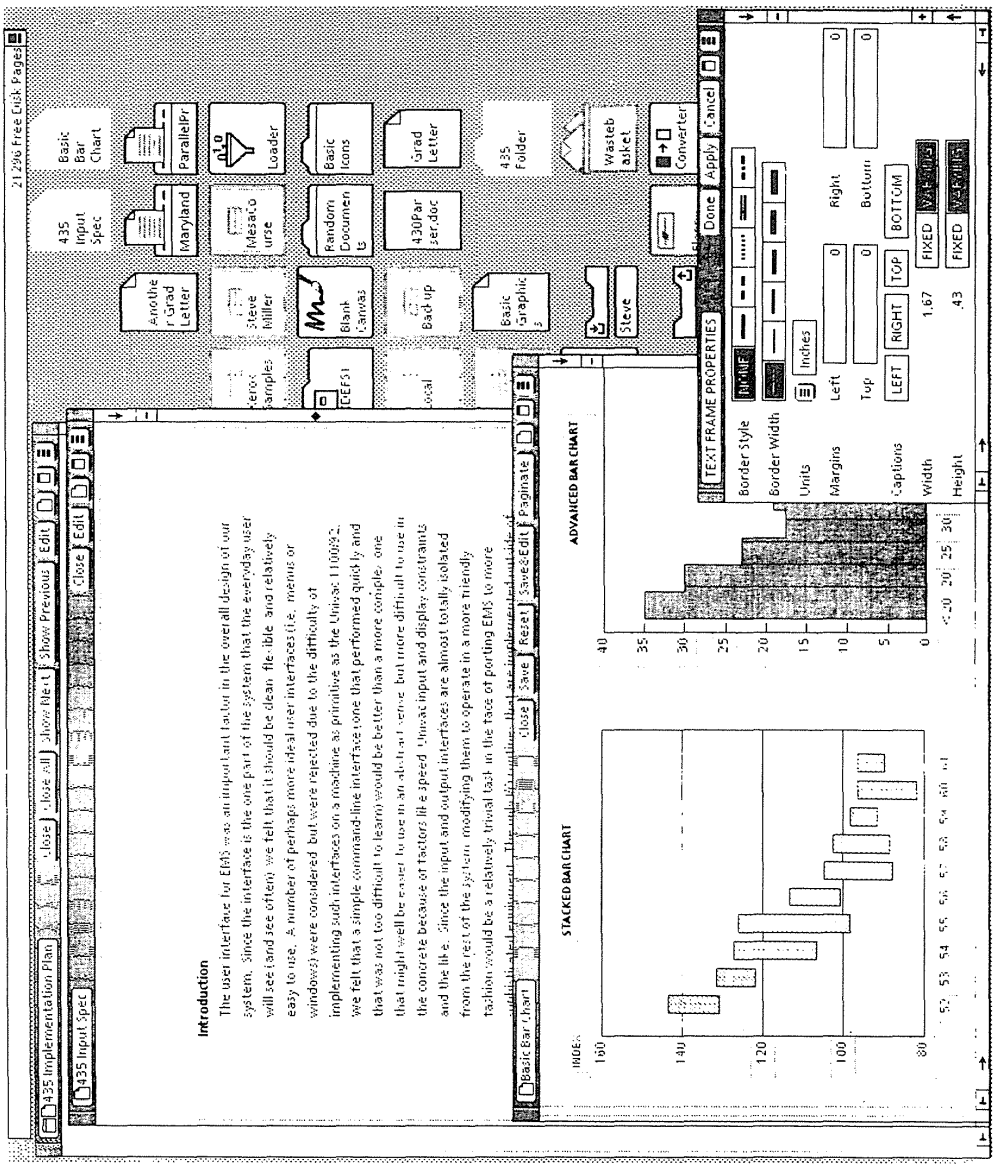


Figure 6.8

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.)

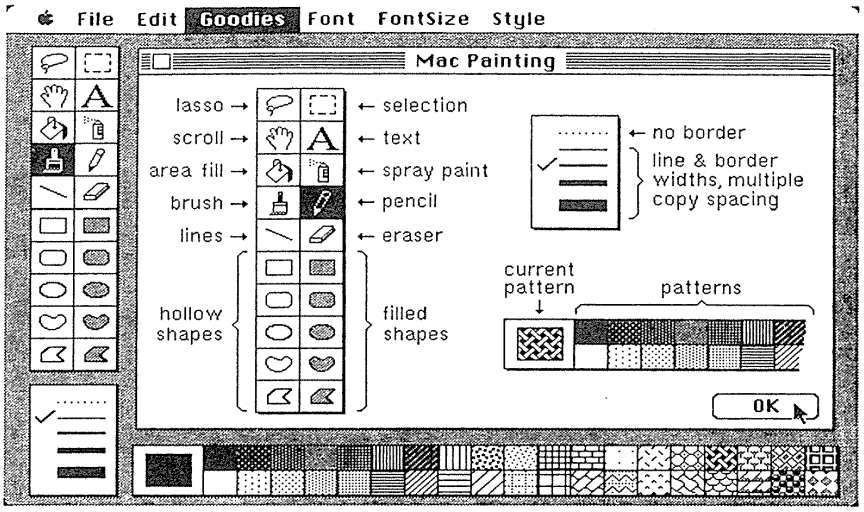


Figure 6.9

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-

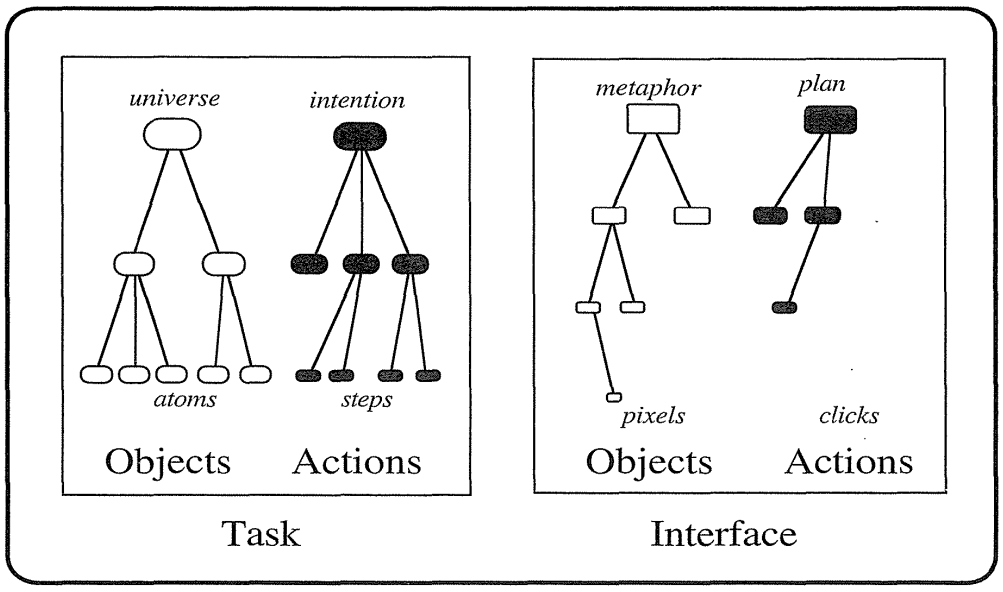


Figure 6.10

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

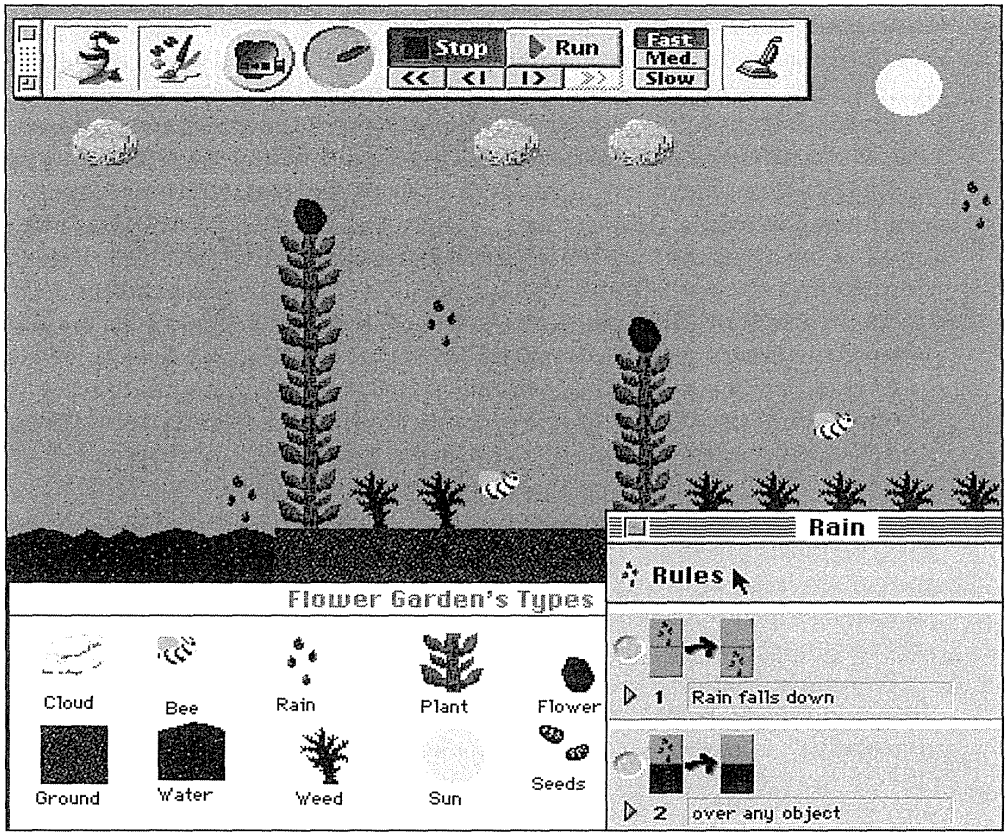


Figure 6.11

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

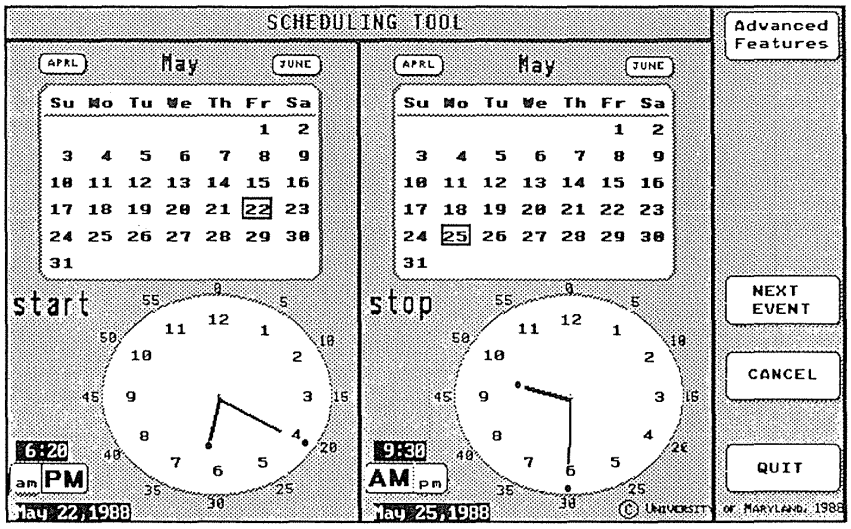


Figure 6.12

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),

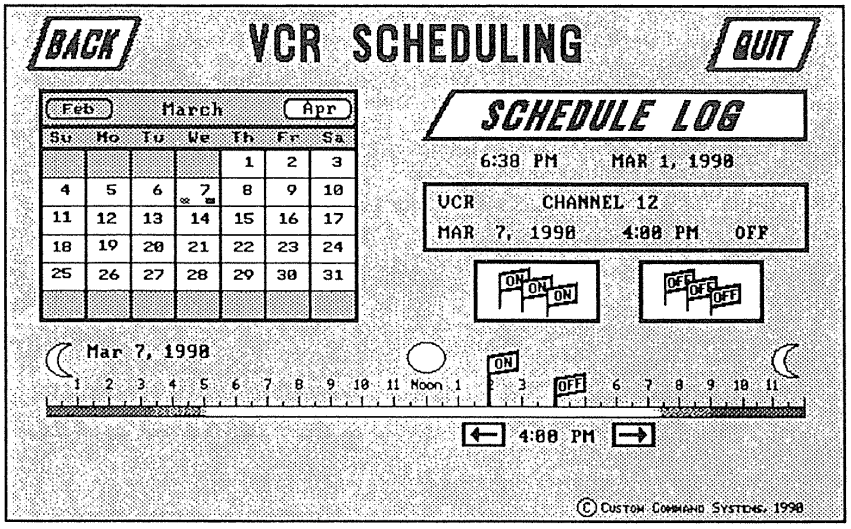


Figure 6.13

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

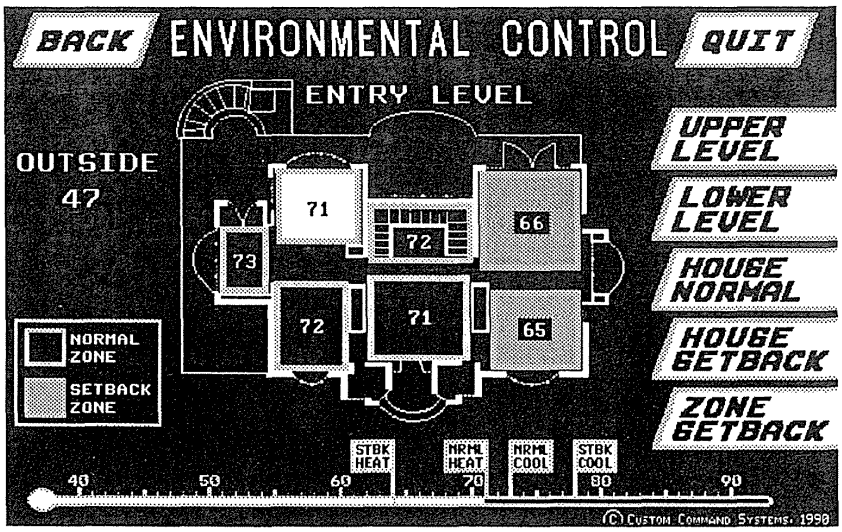


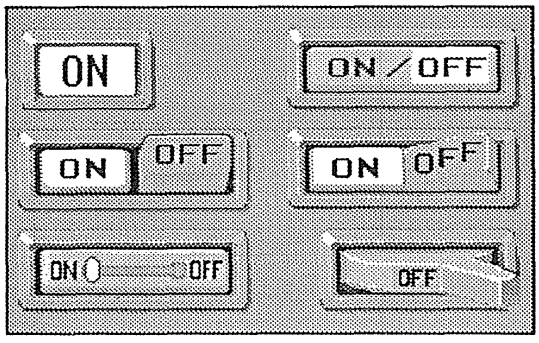
Figure 6.14

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

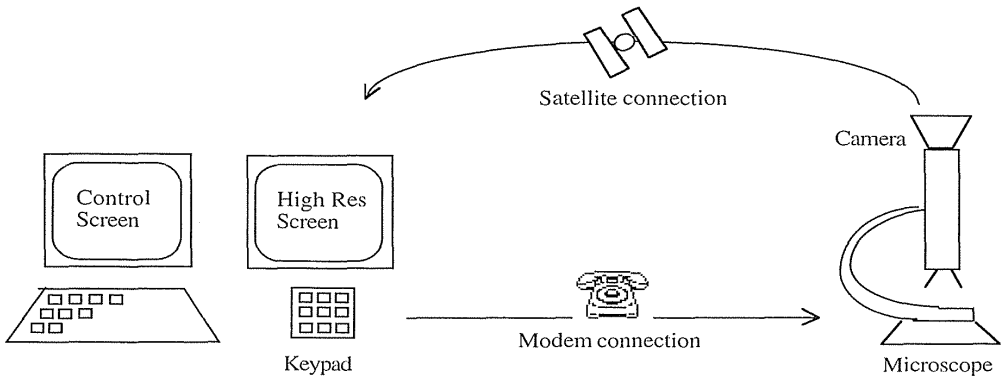


Figure 6.16

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,

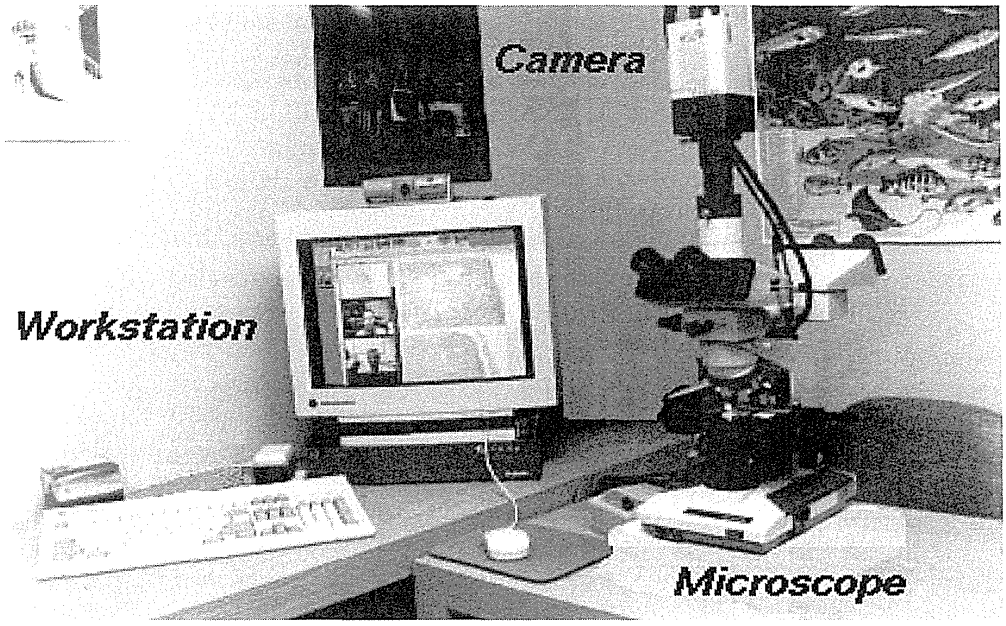


Figure 6.17

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

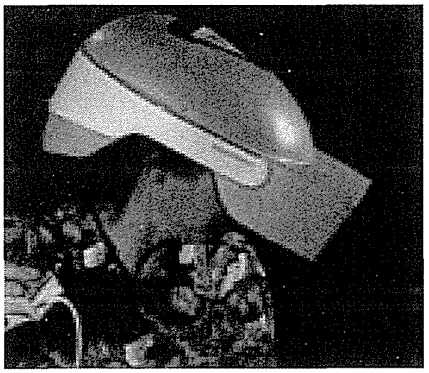
Flight-simulator designers use many tricks to create the most realistic experience for fighter or airline pilots. The cockpit displays and controls are taken from the same production line that create the real ones. Then, the windows are replaced by high-resolution computer displays, and sounds are choreographed 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

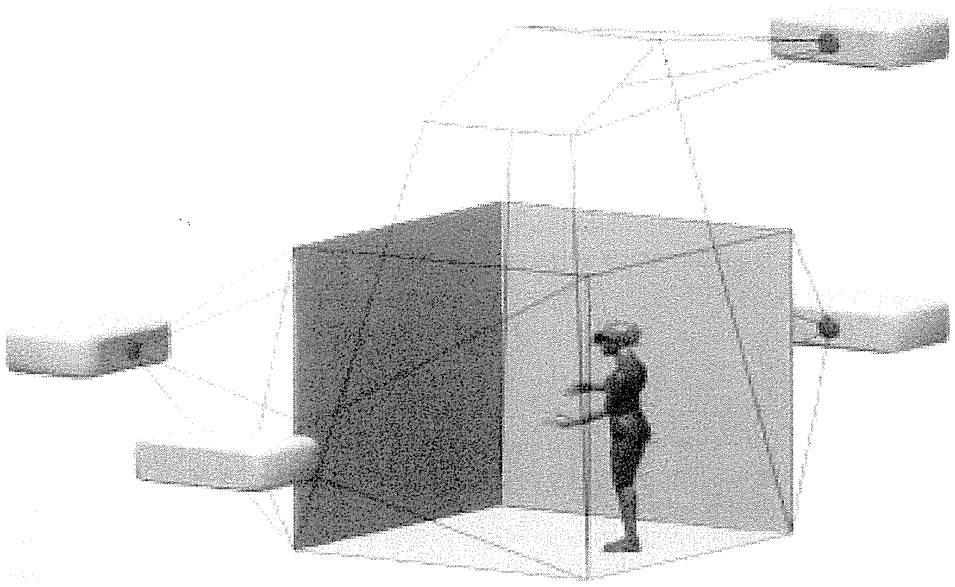


Figure 6.19

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-

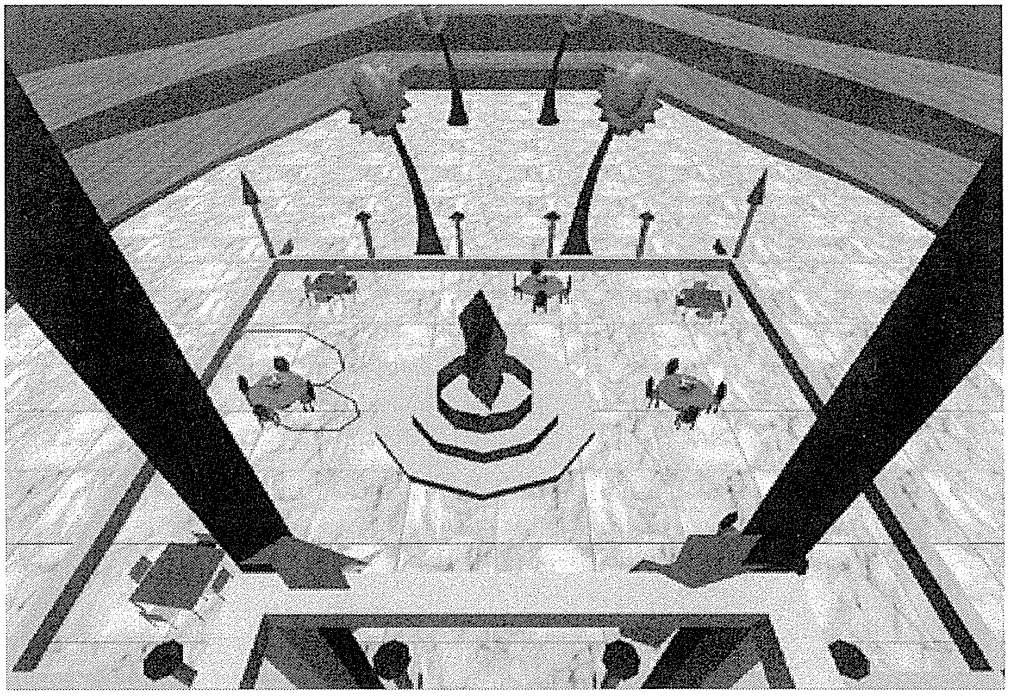


Figure 6.20

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-

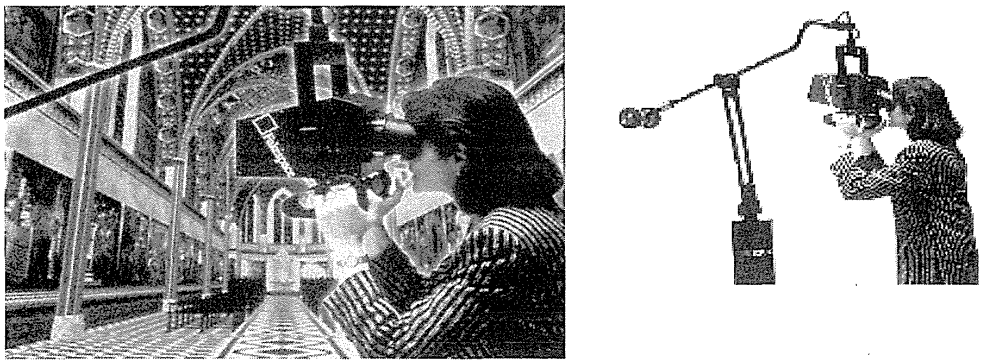


Figure 6.21

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.