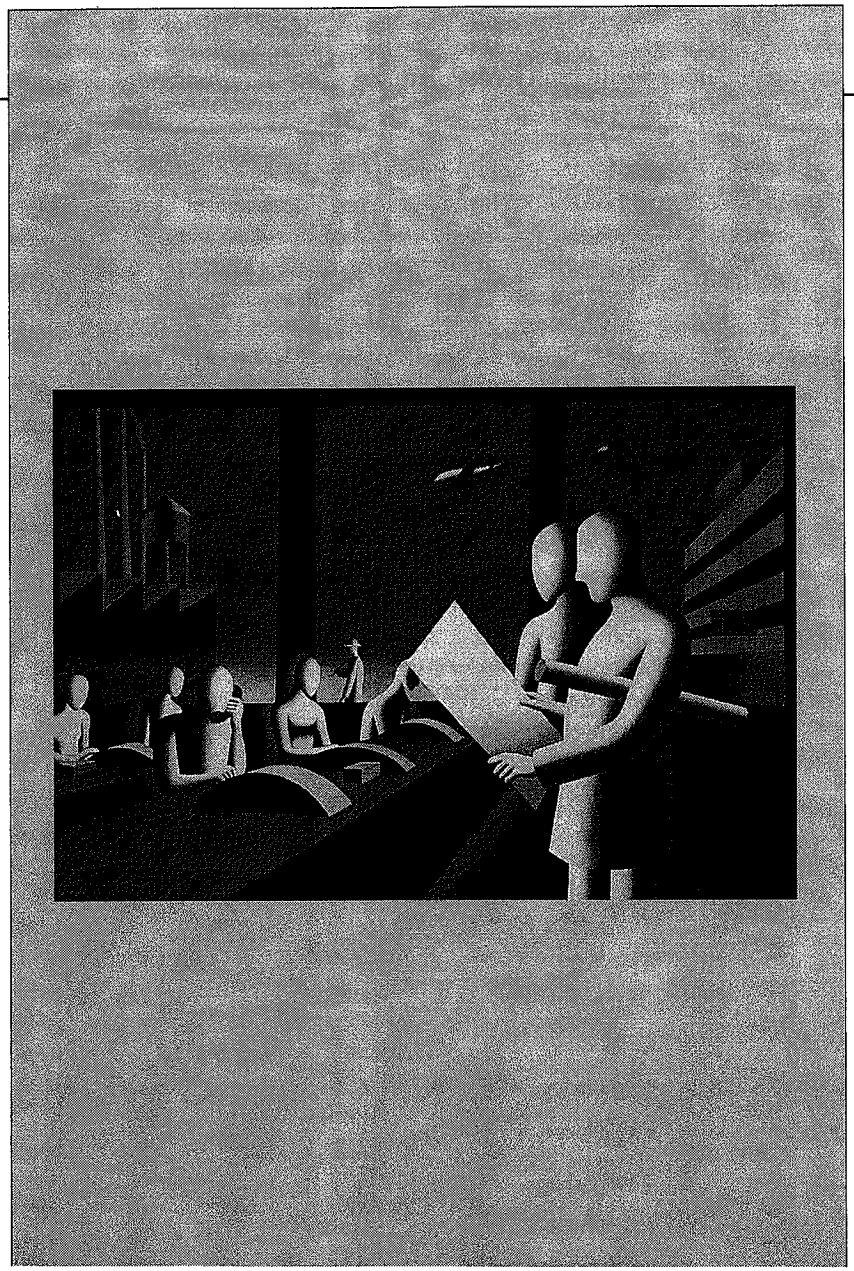


CHAPTER

2

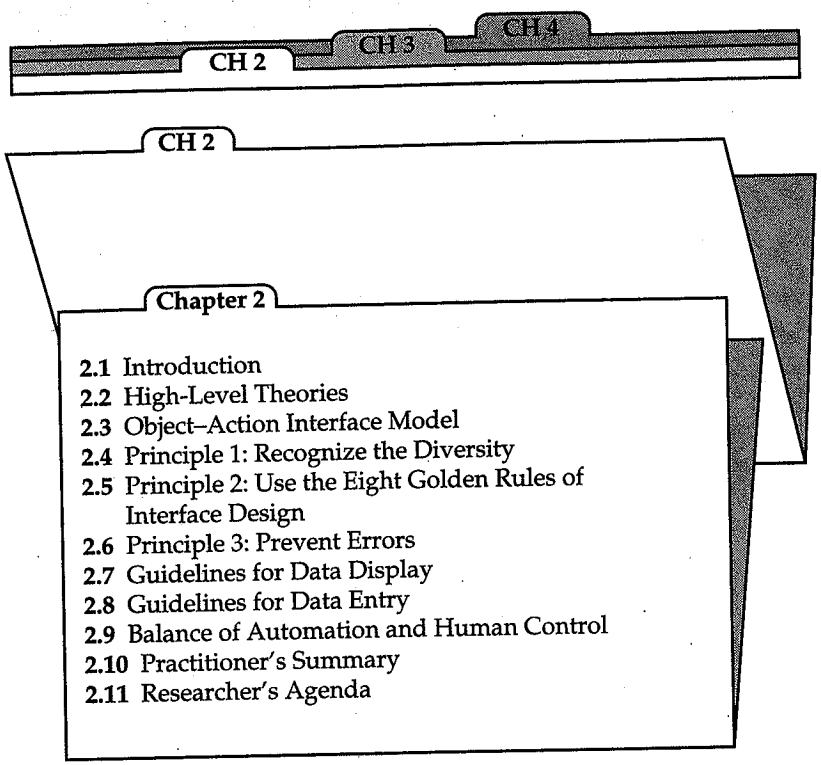
Theories, Principles, and Guidelines

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

Horace Mann, *Thoughts*, 1867

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

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



1 Introduction

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

In many contemporary systems, there is a grand opportunity to improve

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

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

2.2 High-Level Theories

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

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

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

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

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

2.2.1 Conceptual, semantic, syntactic, and lexical model

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

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

This approach is convenient for designers because its top-down nature is easy to explain, matches the software architecture, and allows for useful modularity during design. Designers are expected to move from conceptual to lexical, and to record carefully the mappings between levels.

2.2.2 GOMS and the keystroke-level model

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

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

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

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

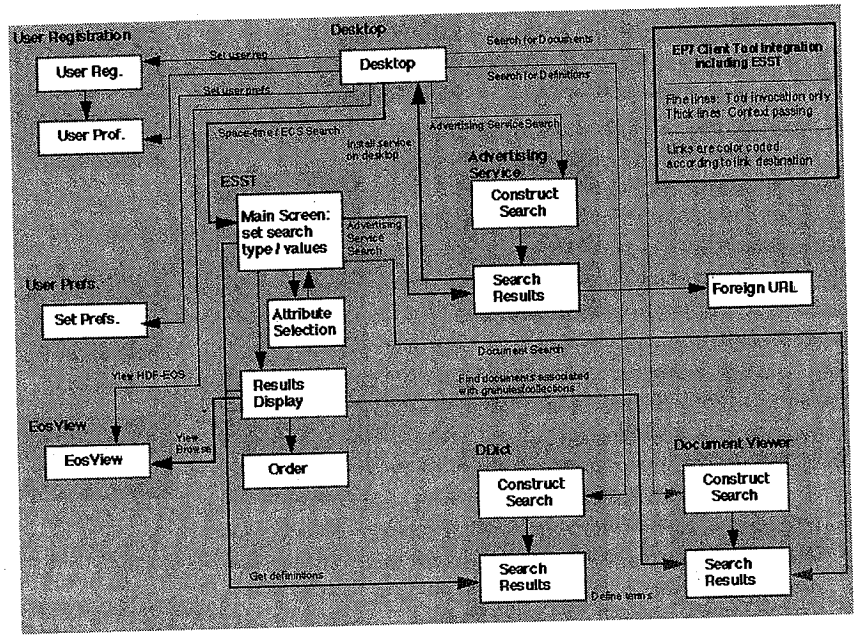


Figure 2.1

Transition diagram from the NASA search system.

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

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

  - Step 5: Report goal accomplished.
- Selection rule set for goal of using a specific field-delete method:
  - If you want to paste the field somewhere else, then choose "Cut Field" from the Edit menu.
  - If you want to delete the field permanently, then choose "Clear Field" from the Edit menu.
  - Report goal accomplished.

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

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

2.2.3 Stages of action models

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

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

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

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

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

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

2.2.4 Consistency through grammars

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

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

Each of the actions in the consistent version is the same, whereas the actions vary for the inconsistent version A. The inconsistent action verbs are all acceptable, but their variety suggests that they will take longer to learn, will cause more errors, will slow down users, and will be harder for users to remember. Inconsistent version B is somehow more malicious because there is a single unpredictable inconsistency that stands out so dramatically that this language is likely to be remembered for its peculiar inconsistency.

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

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

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

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

These schemas will generate a consistent grammar:

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

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

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

2.2.5 Widget-level theories

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

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

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

Gradually, higher-level patterns of usage are appearing, in much that way that Alexander describes has occurred in architecture (1977). Familiar patterns of building fireplaces, stairways, or roofs become modular components that acquire names and are combined to form still larger patterns.

2.3 Object–Action Interface Model

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

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

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

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

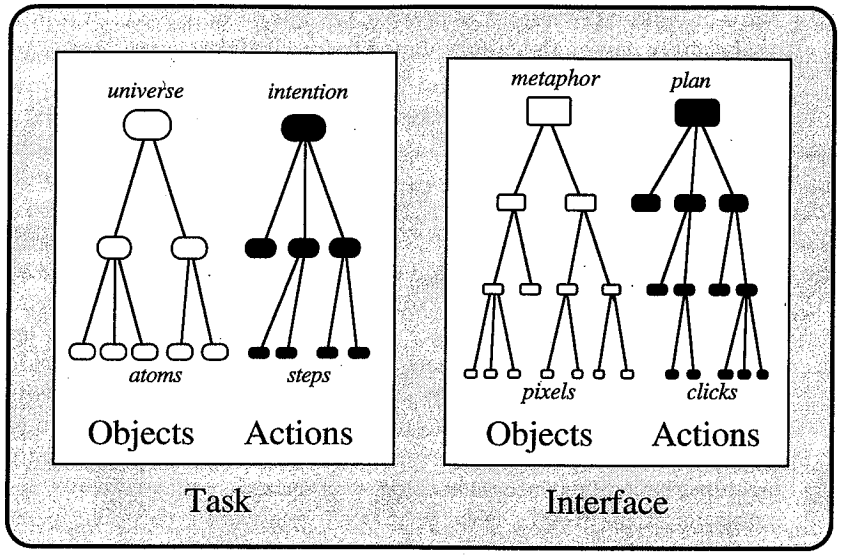


Figure 2.2

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

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

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

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

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

2.3.1 Task hierarchies of objects and actions

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

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

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

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

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

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

2.3.2 Interface hierarchies of objects and actions

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

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

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

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

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

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

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

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

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

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

2.3.3 The disappearance of syntax

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

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

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

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

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

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

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

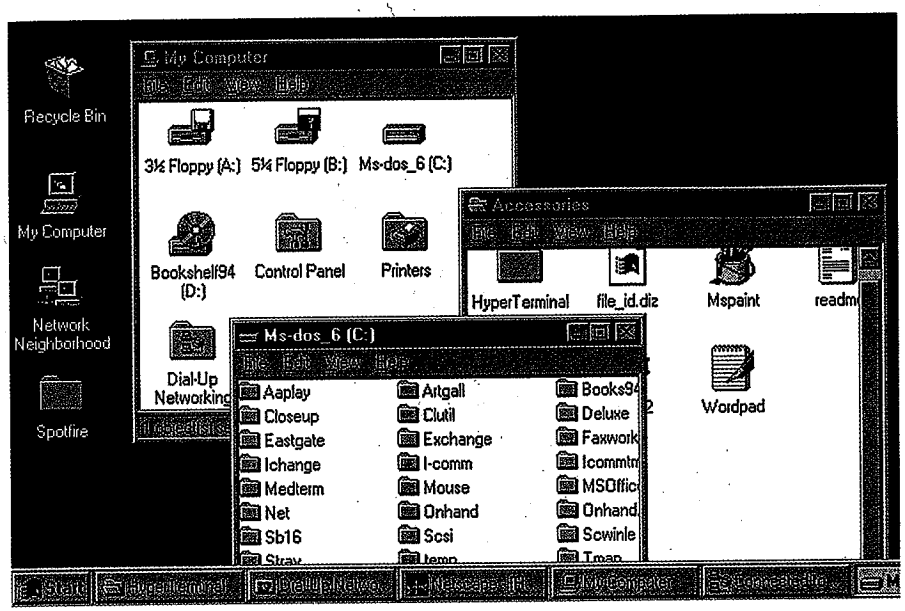


Plate A1: Microsoft Windows 95. (Reprinted with permission from Microsoft Corporation, Redmond, WA.)

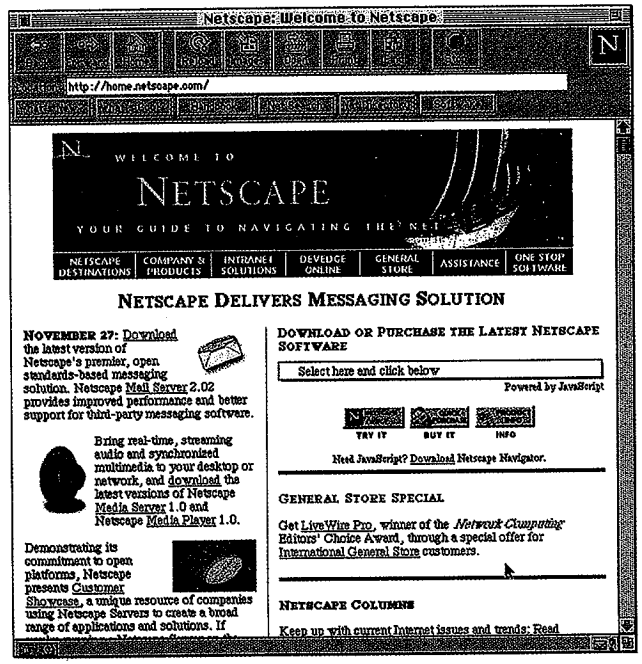


Plate A2: Netscape home page (http://home.netscape.com). (© 1996 Netscape Communication Corporation. Used with permission.)

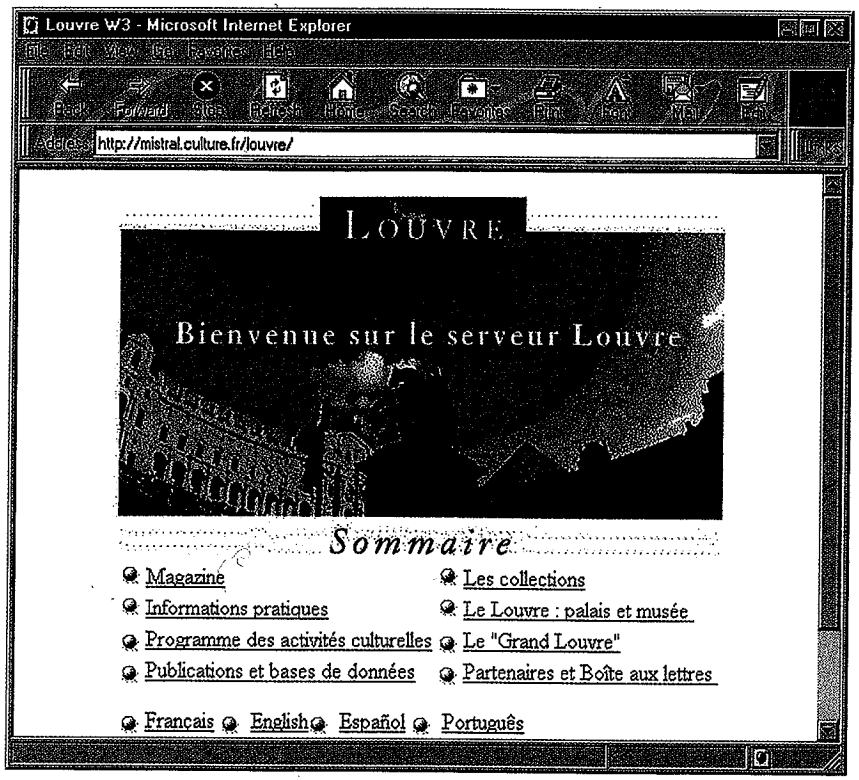


Plate A3: Microsoft Internet Explorer, showing the Louvre Museum web site (http://mistral.culture.fr/louvre).

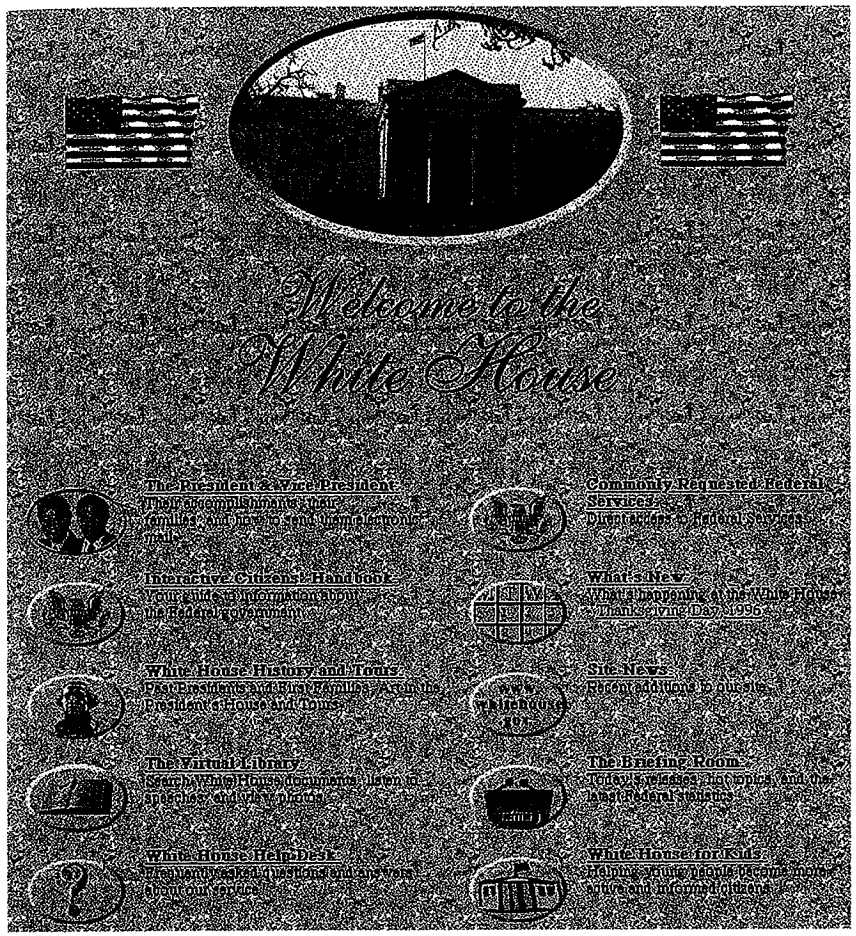


Plate A4: White House home page (http://www.whitehouse.gov).

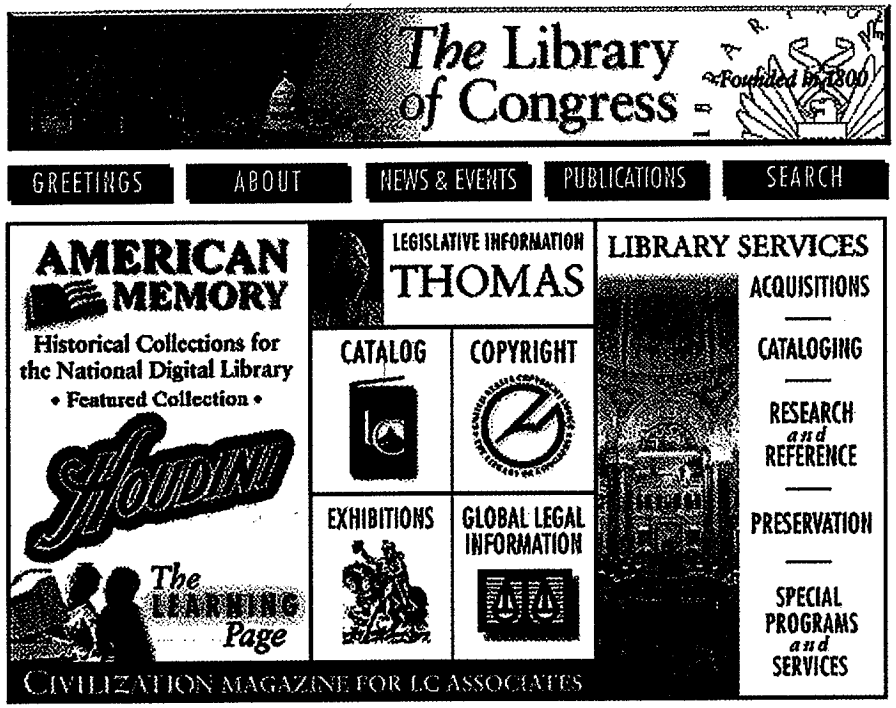


Plate A5: Library of Congress home page (http://www.loc.gov).

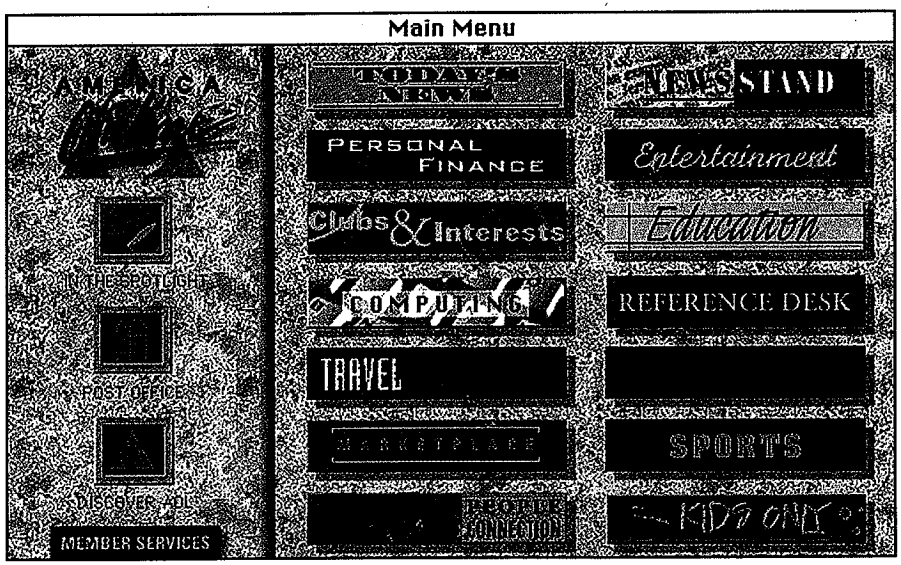


Plate A6: America Online main menu. (© 1997 America Online. Used by permission.)

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

2.4 Principle 1: Recognize the Diversity

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

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

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

2.4.1 Usage profiles

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

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

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

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

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

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

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

- *Novice or first-time users* True novice users are assumed to know little of the task or interface concepts. By contrast, first-time users are professionals who know the task concepts, but have shallow knowledge of the interface concepts. Both groups of users may arrive with anxiety about using computers that inhibits learning. Overcoming these limitations is a serious challenge to the designer of the interface, including instructions, dialog boxes, and online help. Restricting vocabulary to a small number of familiar, consistently used concept terms is essential to begin developing the user's knowledge. The number of actions should also be small, so that novice and first-time users can carry out simple tasks successfully and thus reduce anxiety, build confidence, and gain positive reinforcement. Informative feedback about the accomplishment of each task is helpful, and constructive, specific error messages should be provided when users make mistakes. Carefully designed paper manuals and step-by-step online tutorials may be effective.
- *Knowledgeable intermittent users* Many people are knowledgeable but intermittent users of a variety of systems. They have stable task concepts and broad knowledge of interface concepts, but they will have difficulty retaining the structure of menus or the location of features. The burden on their memories will be lightened by orderly structure in the menus, consistent terminology, and high interface apparency, which emphasizes recognition rather than recall. Consistent sequences of actions, meaningful messages, and guides to frequent patterns of usage will help knowledgeable intermittent users to rediscover how to perform their tasks properly. Protection from danger is necessary to support relaxed exploration of features or attempts to invoke a partially forgotten action sequence. These users will benefit from online help screens to fill in missing pieces of task or interface knowledge. Well-organized reference manuals also will be useful.
- *Expert frequent users* Expert "power" users are thoroughly familiar with the task and interface concepts and seek to get their work done quickly. They demand rapid response times, brief and nondistracting feedback, and the capacity to carry out actions with just a few keystrokes or selections. When a sequence of three or four commands is performed regularly, the frequent user is eager to create a *macro* or other abbreviated form to reduce the number of steps. Strings of commands, shortcuts through menus, abbreviations, and other accelerators are requirements.

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

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

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

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

2.4.2 Task profiles

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

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

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

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

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

FREQUENCY OF TASK BY JOB TITLE

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

Figure 2.3

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

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

2.4.3 Interaction styles

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

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

Box 2.1

Advantages and disadvantages of the five primary interaction styles.

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

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

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

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

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

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

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

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

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

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

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

1. *Strive for consistency.* This rule is the most frequently violated one, but following it can be tricky because there are many forms of consistency. Consistent sequences of actions should be required in similar situations; identical terminology should be used in prompts, menus, and help screens; and consistent color, layout, capitalization, fonts, and so on should be employed throughout. Exceptions, such as no echoing of passwords or confirmation of the delete command, should be comprehensible and limited in number.
2. *Enable frequent users to use shortcuts.* As the frequency of use increases, so do the user's desires to reduce the number of interactions and to increase the pace of interaction. Abbreviations, special keys, hidden commands, and macro facilities are appreciated by frequent knowledgeable users. Short response times and fast display rates are other attractions for frequent users.
3. *Offer informative feedback.* For every user action, there should be system feedback. For frequent and minor actions, the response can be modest, whereas for infrequent and major actions, the response should be more substantial. Visual presentation of the objects of interest provides a convenient environment for showing changes explicitly (see discussion of direct manipulation in Chapter 6).
4. *Design dialogs to yield closure.* Sequences of actions should be organized into groups with a beginning, middle, and end. The informative feedback at the completion of a group of actions gives operators the satisfaction of accomplishment, a sense of relief, the signal to drop contingency plans and options from their minds, and an indication that the way is clear to prepare for the next group of actions.
5. *Offer error prevention and simple error handling.* As much as possible, design the system such that users cannot make a serious error; for example, prefer menu selection to form fillin and do not allow alphabetic characters in numeric entry fields. If users make an error, the system should detect the error and offer simple, constructive, and specific instructions for recovery. For example, users should not have to retype an entire command, but rather should need to repair only the faulty part. Erroneous actions should leave the system state unchanged, or the system should give instructions about restoring the state.
6. *Permit easy reversal of actions.* As much as possible, actions should be reversible. This feature relieves anxiety, since the user knows that errors can be undone, thus encouraging exploration of unfamiliar options. The units of reversibility may be a single action, a data-entry task, or a complete group of actions such as entry of a name and address block.
7. *Support internal locus of control.* Experienced operators strongly desire the sense that they are in charge of the system and that the system responds to their actions. Surprising system actions, tedious sequences of data entries, inability or difficulty in obtaining necessary information, and inability to produce the action desired all build anxiety and dissatisfaction. Gaines (1981) captured part of this principle with his rule *avoid acausality* and his encouragement to make users the *initiators* of actions rather than the *responders* to actions.
8. *Reduce short-term memory load.* The limitation of human information processing in short-term memory (the rule of thumb is that humans can remember "seven-plus or minus-two chunks" of information) requires that displays be kept simple, multiple page displays be consolidated, window-motion frequency be reduced, and sufficient training time be allotted for codes, mnemonics, and sequences of actions. Where appropriate, online access to command-syntax forms, abbreviations, codes, and other information should be provided.

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

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

2.6 Principle 3: Prevent Errors

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

Sigmund Freud (Inscription he wrote on his portrait)

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

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

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

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

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

2.6.1 Correct matching pairs

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

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

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

2.6.2 Complete sequences

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

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

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

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

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

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

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

2.6.3 Correct commands

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

The same principles can be applied to interactive systems. Consider these typical errors made by the users of command languages: They invoke commands that are not available, request files that do not exist, or enter data values that are not acceptable. These errors are often caused by annoying typographic errors, such as using an incorrect command abbreviation; pressing a pair of keys, rather than a desired single key; misspelling a file name; or making a minor error such as omitting, inserting, or transposing characters. Error messages range from the annoyingly brief ? or WHAT?, to the vague UNRECOGNIZED COMMAND or SYNTAX ERROR, to the condemning BAD FILE NAME or ILLEGAL COMMAND. The brief ? is suitable for expert users who have made a trivial error and can recognize it when they see the command line on the screen. But if an expert has ventured to use a new command and has misunderstood its operation, then the brief message is not helpful. They must interrupt their planning to deal with correcting the problem—and with their frustration in not getting what they wanted.

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

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

2.7 Guidelines for Data Display

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

2.7.1 Organizing the display

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

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

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

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

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

Chapter 11 further discusses data-display issues.

2.7.2 Getting the user's attention

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

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

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

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

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

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

2.8 Guidelines for Data Entry

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

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

   A second aspect of this guideline is that redundant data entry should be avoided. It is annoying for users to enter the same information in two locations, since the double entry is perceived as a waste of effort and an opportunity for error. When the same information is required in two places, the system should copy the information for the user, who still has the option of overriding by retyping.
3. *Minimal memory load on users* When doing data entry, users should not be required to remember lengthy lists of codes and complex syntactic command strings.
4. *Compatibility of data entry with data display* The format of data-entry information should be linked closely to the format of displayed information.
5. *Flexibility for user control of data entry* Experienced data-entry operators may prefer to enter information in a sequence that they can control. For example, on some occasions in an air-traffic control environment, the arrival time is the prime field in the controller's mind; on other occasions, the altitude is the prime field. Flexibility should be used cautiously, since it goes against the consistency principle.

2.9 Balance of Automation and Human Control

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

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

Box 2.2

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

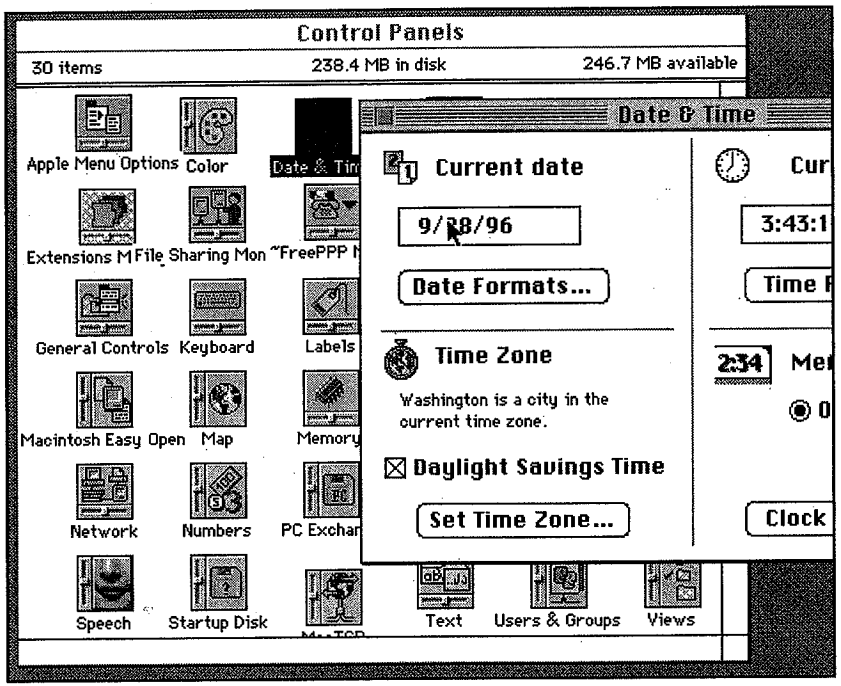


Figure 2.4

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

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

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

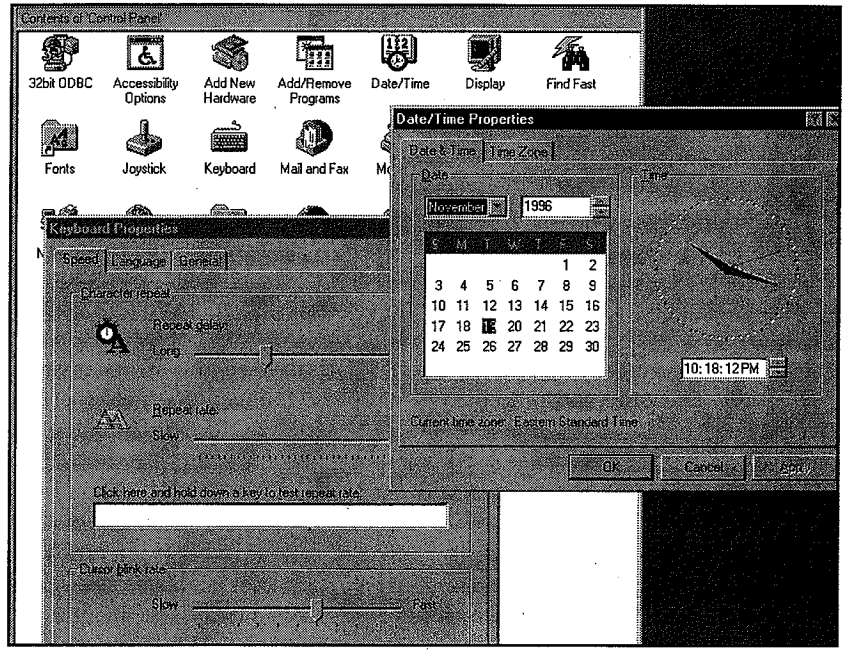


Figure 2.5

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

2.10 Practitioner's Summary

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

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

2.11 Researcher's Agenda

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

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

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

World Wide Web Resources WWW

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

http://www.aw.com/DTUI

References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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