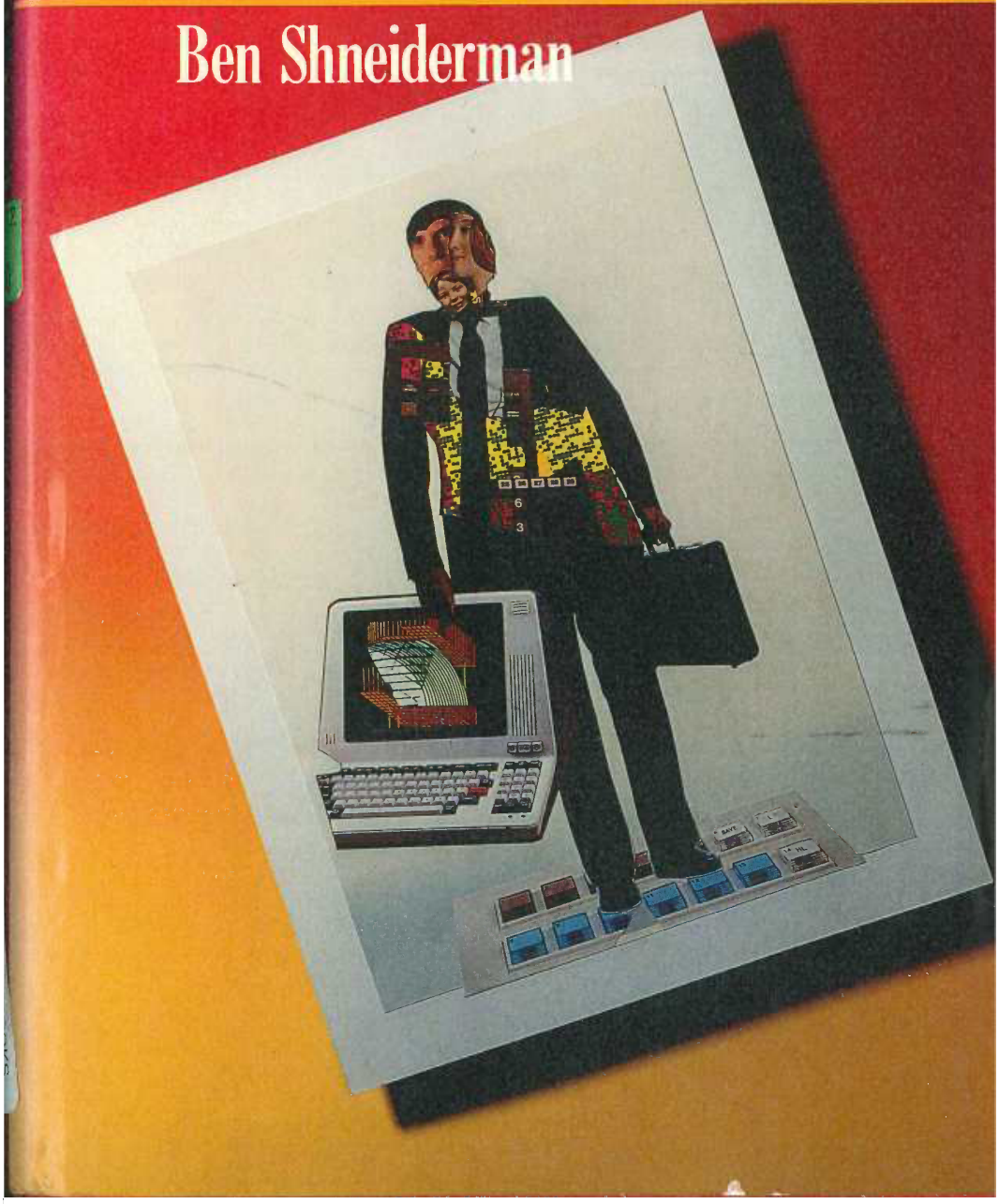
Designing the User Interface

Strategies for Effective Human-Computer Interactio

Ben Shneiderman



Designing the User I
ace
Strategies for Effective
Human-Computer Interaction
Shneiderman
12/29/12
$ 3.00
760
THRIFTBOOKS
11 05414 760

Designing the User Interface

Strategies for Effective Human-Computer Interaction

Ben Shneiderman
University of Maryland

This book presents a comprehensive survey of the specifically *human* factors that need to be considered in designing interactive computer systems. System designers, programmers, and product managers increasingly recognize the vital importance of the human-computer, or user, interface for the success of their systems. They can find here a carefully researched and clearly written discussion of the issues and principles necessary to understand these human factors, and the practical techniques necessary to realize an effective design.

Author Ben Shneiderman makes a strong case in this book for giving the user power over the system. He has too often seen the system controlling the user and here he explores the problems that can arise when this happens. His aim, ultimately, is to make computer systems truly powerful yet comprehensible, and he provides ample direction in *Designing the User Interface* to help make it happen.

The book is divided into four parts:

- Motivations and Foundations
- Interaction Styles: menu selection systems, command languages, and direct manipulation
- Considerations and Augmentations: interaction devices, response time and display rate, screen design, color, windows, and user manuals
- Assessment and Reflection: testing and evaluation

It includes references at the end of each chapter for readers interested in further information on the topics presented.

Designing the User Interface: Strategies for Effective Human-Computer Interaction is written for anyone who needs to understand the complex interaction of people and machines. It will benefit both professionals and students, people who build the interactive systems and people who market them, people who research system design and people who research human performance. Readers will understand, in the end, why systems designed with human factors clearly in mind will have the competitive edge in information retrieval, office automation, and personal computing markets.

ADDISON-WESLEY PUBLISHING COMPANY

ISBN 0-201-16505-8

Designing the User Interface: Strategies for Effective Human-Computer Interaction

BEN SHNEIDERMAN
The University of Maryland

Illustrations by Carol Wald

ADDISON-WESLEY PUBLISHING COMPANY
READING, MASSACHUSETTS • MENLO PARK, CALIFORNIA
DON MILLS, ONTARIO • WOKINGHAM, ENGLAND • AMSTERDAM
SYDNEY • SINGAPORE • TOKYO • MADRID • BOGOTÁ
SANTIAGO • SAN JUAN

Library of Congress Cataloging-in-Publication Data
Shneiderman, Ben.
Designing the User Interface: Strategies for Effective Human-Computer Interaction

Bibliography: p.
Includes indexes.
1. Interactive computer systems. 2. System design.
I. Title.
QA76.9.I58S47 1987 004.6 85–28765
ISBN 0–201–16505–8

Reprinted with corrections May, 1987

Text reproduced by Addision-Wesley from camera-ready materials provided by the author. Cover and chapter opening illustrations ©Carol Wald.

FGHIJ-HA–89

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

2.1 INTRODUCTION

Successful designers of interactive systems know that they can and must go beyond intuitive judgments made hastily when a design problem emerges. Fortunately, guidance for designers is beginning to emerge in the form of:

- high-level theories or models
- middle-level principles
- specific and practical guidelines and
- strategies for testing.

The theories or models offer a way to organize the design process, whereas the middle-level principles are useful in weighing alternatives. The practical guidelines provide helpful reminders of rules uncovered by previous designers. Early prototype evaluation encourages exploration and enables iterative design to correct inappropriate decisions. Acceptance testing is the trial-by-fire to determine whether a system is ready for distribution; its presence may be seen as a challenge, but it is also a gift to designers since it establishes clear measures of success.

In many contemporary systems, there is a grand opportunity to improve the human interface. The cluttered and multiple displays, complex and tedious procedures, inadequate command languages, inconsistent sequences of actions, and insufficient informative feedback can generate debilitating stress and anxiety that lead to poor performance, frequent minor and occasional serious errors, and job dissatisfaction.

This chapter begins with a review of theories, concentrating on the syntactic/semantic model. Section 2.3 then deals with frequency of use, task profiles, and interaction styles. Eight principles of interaction are offered in Section 2.4. Strategies for preventing errors are described in Section 2.5. Specific guidelines for data entry and display appear in Sections 2.6 and 2.7. Testing strategies are presented in Section 2.8 and covered in detail in Chapter 10. Section 2.9 attempts to deal with the difficult question of the balance between automation and human control.

2.2 A HIGH-LEVEL THEORY: SYNTACTIC/ SEMANTIC MODEL OF USER KNOWLEDGE

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. Interactive system designers can benefit from a syntactic/semantic model of user knowledge. In outline, this explanatory model suggests that users have syntactic knowledge about device-dependent details and semantic knowledge about concepts. The semantic knowledge is separated into task concepts (objects and actions) and computer concepts (objects and actions) (see Figure 2.1). A person can be an expert in the computer concepts, but a novice in the task, and vice versa.

The syntactic/semantic model of user behavior was originated to describe programming (Shneiderman & Mayer, 1979; Shneiderman,

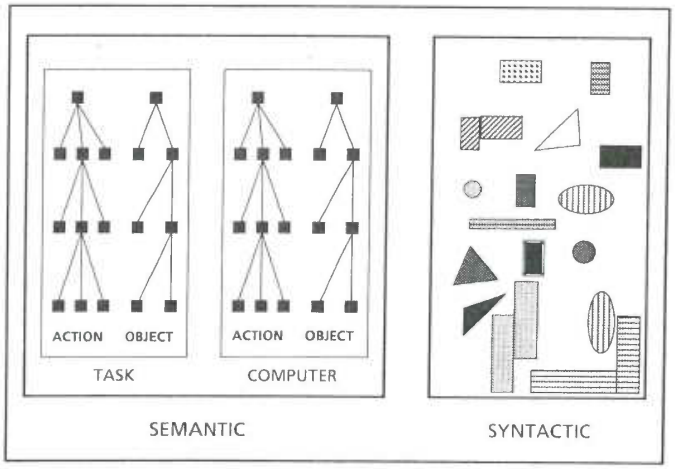
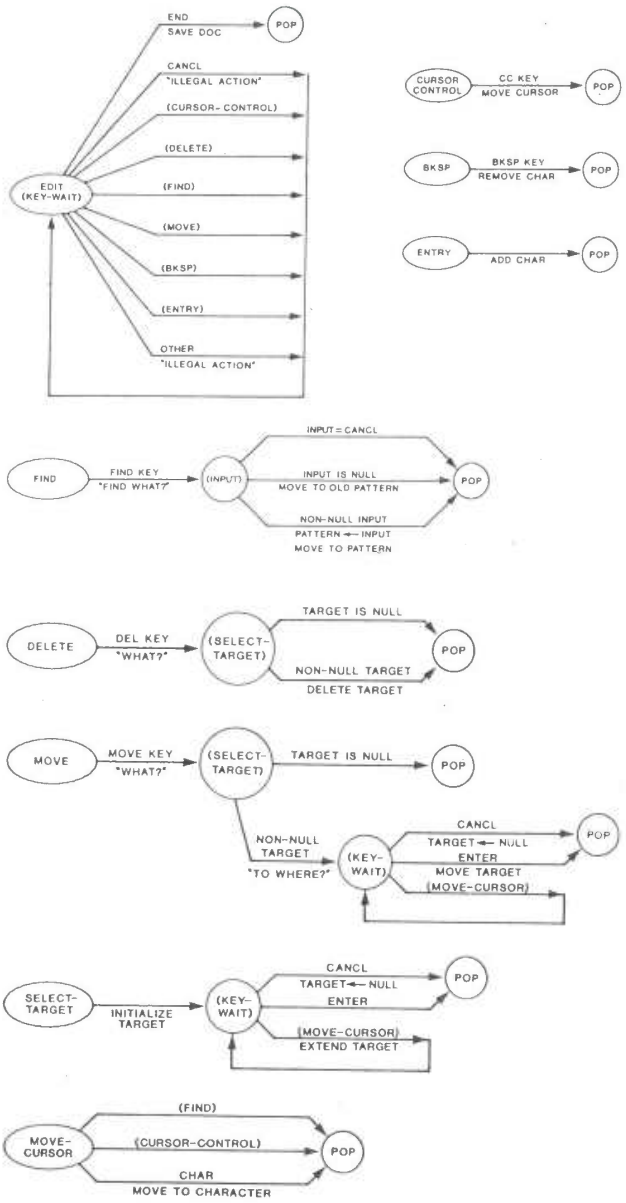


Figure 2.1: A representation of the user's knowledge in long-term memory. The syntactic knowledge is varied, device dependent, acquired by rote memorization, and easily forgotten. The semantic knowledge is separated into the computer and task domains. Within these domains, knowledge is divided into actions and objects. Semantic knowledge is structured, device independent, acquired by meaningful learning, and stable in memory.


EDIT (KEY-WAIT)
END
SAVE DOC
POP
CANCL
"ILLEGAL ACTION"
(CURSOR-CONTROL)
(DELETE)
(FIND)
(MOVE)
(BKSP)
(ENTRY)
OTHER
"ILLEGAL ACTION"
CURSOR CONTROL
CC KEY
MOVE CURSOR
POP
BKSP
BKSP KEY
REMOVE CHAR
POP
ENTRY
ADD CHAR
POP
FIND
FIND KEY
"FIND WHAT?"
(INPUT)
INPUT = CANCL
INPUT IS NULL
MOVE TO OLD PATTERN
NON-NULL INPUT
PATTERN ← INPUT
MOVE TO PATTERN
POP
DELETE
DEL KEY
"WHAT?"
(SELECT-TARGET)
TARGET IS NULL
NON-NULL TARGET
DELETE TARGET
POP
MOVE
MOVE KEY
"WHAT?"
(SELECT-TARGET)
TARGET IS NULL
POP
NON-NULL TARGET
"TO WHERE?"
(KEY-WAIT)
CANCL
TARGET ← NULL
ENTER
MOVE TARGET
(MOVE-CURSOR)
POP
SELECT-TARGET
INITIALIZE TARGET
(KEY-WAIT)
CANCL
TARGET ← NULL
ENTER
(MOVE-CURSOR)
EXTEND TARGET
POP
MOVE-CURSOR
(FIND)
(CURSOR-CONTROL)
CHAR
MOVE TO CHARACTER
POP

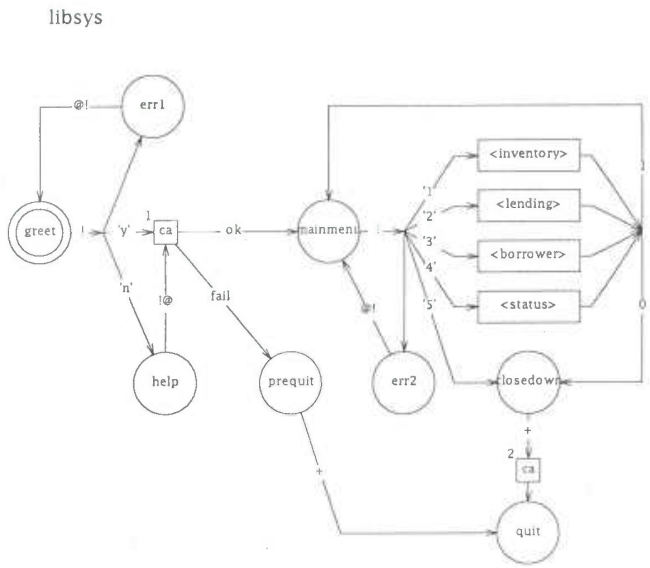


Figure 2.2b: A second example of a transition diagram indicates user actions on the arrows and computer responses or states in the boxes and circles. (Source: Transition diagram editor from Interactive Development)

1980) and has been applied to database manipulation facilities (Shneiderman, 1981) as well as to direct manipulation (Shneiderman, 1983).

Other strategies for modeling interactive system usage involve transition diagrams (Kieras & Polson, 1985) (Figure 2.2). These diagrams are helpful during design, for instruction, and as a predictor of learning time, performance time, and errors.

Figure 2.2a: This generalized transition network for the Displaywriter shows the sequence of permissible actions. If the users begin at the EDIT state and issue a FIND command, they follow the paths in the FIND subdiagram. (Kieras and Polson, "An approach to the formal analysis of user complexity," *International Journal of Man-Machine Studies 22* [1985], 365–394. Used by permission of Academic Press Inc. [London] Limited.)

Other theories include the four-level approach of Foley and Van Dam (1982):

1. The highest level is the conceptual model or 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 syntax 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 the precise mechanism by which a user specifies the syntax.

Card, Moran, and Newell (1980, 1983) proposed the GOMS (goals, operators, methods, and selection rules) model and the keystroke-level model. They postulated that users formulate goals and subgoals that are achieved by methods or procedures for accomplishing each goal. 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, Moran & Newell, 1983: 144). The selection rules are the control structure for choosing among the several methods available for accomplishing a goal.

The keystroke-level model is an attempt 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, with less emphasis on learning, problem-solving, error handling, subjective satisfaction, and retention.

Kieras and Polson (1985) 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.

Norman (1984) offers a stages and levels model of human-computer interaction. The four stages of an interaction are:

1. Forming an intention: internal mental characterization of a goal.
2. Selecting an action: review possible actions and select most appropriate.
3. Executing the action: carry out the action using the computer.
4. Evaluating the outcome: check the results of executing the action.

Norman's levels 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, and eventually produces the action of moving the mouse to select a point on the screen. Norman describes how errors often occur as intentions are reformulated to lower levels.

Refinements to these theories and more detailed models will undoubtedly emerge. There is a need for many theories about the multiple aspects of interactive systems. The next section expands on the syntactic/semantic model of user behavior.

2.2.1 Syntactic knowledge

When using a computer system, users must maintain a profusion of device-dependent details in their human memory. These low-level syntactic details include the knowledge of which key erases a character (delete, backspace, CTRL-H, rightmost mouse button, or ESCAPE), what command inserts a new line after the third line of a text file (CTRL-I, INSERT key, `I3`, `I 3`, or `3I`), which icon to click on to scroll text forward, which abbreviations are permissible, and which of the numbered function keys produces the previous screen.

The learning, use, and retention of this knowledge is 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 frequently applied. Syntactic knowledge is usually conveyed by example and repeated usage. Formal notations, such as Backus-Naur Form, are useful for knowledgeable computer scientists but confusing to most users.

A further problem with syntactic knowledge, in some cases, is 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. To be sure, there may be some overlap. For example, arithmetic expressions might be the same in two languages; but unfortunately, 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 intermittent users, however, are especially troubled by syntactic irregularities. Their burden can be lightened by using menus (see Chapter 3), 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 4).

In summary, syntactic knowledge is arbitrary, system dependent, and ill-structured. It must be acquired by rote memorization and repetition. Unless it is regularly used, it fades from memory.

2.2.2 Semantic knowledge—computer concepts

Semantic knowledge in human long-term memory has two components: computer concepts and task concepts (Figure 2.1). Semantic knowledge has a hierarchical structure ranging from low-level actions to middle-level strategies to high-level goals (Shneiderman & Mayer, 1979; Soloway et al., 1982; Card et al., 1983). This presentation enhances the earlier syntactic/semantic model and other models by decoupling computer concepts from task concepts. This enhancement accommodates the two most common forms of expertness: task experts who may be novice computer users and computer user experts who may be new to a task. Different training materials are suggested for task or computer experts. Novices in both domains need yet a third form of training.

Semantic knowledge is conveyed by showing examples of use, offering a general theory or pattern, relating the concepts to previous knowledge by analogy, describing a concrete or abstract model, and by indicating examples of incorrect use. There is an attraction to showing incorrect use to indicate clearly the bounds of a concept, but there is also a danger since the learner may confuse correct and incorrect use. Pictures are often helpful in showing the relationships among semantic knowledge concepts.

Computer concepts include *objects* and *actions* at high and low levels. For example, a central set of *computer object* concepts deals with storage. Users come to understand the high-level concept that computers store information. The concept of stored information can be refined into the object concepts of the directory and the files of information. In turn, the directory object is refined into a set of directory entries that each have 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, pointers, binary numbers, and so on.

The *computer actions* are also decomposable into lower level actions. The high-level actions or goals, such as creating a text data file, may require load, insertion, and save actions. The mid-level 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, there is the low-level action of issuing a specific command, carried out by the syntactic detail of pressing the RETURN key.

These computer concepts were designed by highly trained experts in the hope that they were logical, or at least "made sense" to the designers. Unfortunately, the logic may be a very complex product of underlying hardware, software, or performance constraints, or it might be just poorly chosen. Users are often confronted with computer concepts that they have great difficulty absorbing; but hopefully, designers are improving and computer literacy training is raising knowledge levels. For example, the action of terminating a command by pressing RETURN is more and more widely known, even by nonprogrammers.

Users can learn computer concepts by seeing a demonstration of commands, hearing an explanation of features, or by trial and error. A common practice is to create a model of concepts, either abstract, concrete, or analogical, to convey the computer action. For example, with the file-saving concept, 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 make a library analogy and describe how the card catalog acts as a directory for books saved in the library.

Since semantic knowledge about computer concepts has a logical structure and since it can be anchored to familiar concepts, this knowledge is expected to be relatively stable in memory. If you remember the high-level concept about saving a file, you 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 potential for a visual presentation support the memorability of this knowledge.

These computer concepts were once novel and known to 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 computer concepts.

In summary, the user must acquire semantic knowledge about computer concepts. These concepts are hierarchically organized, acquired by meaningful learning or analogy, independent of the syntactic details, hopefully transferable across different computer systems, and relatively stable in memory. Computer concepts can be usefully sorted out into objects and actions.

2.2.3 Semantic knowledge—task concepts

The primary way for people to deal with large and complex problems is to decompose them into several smaller problems in a hierarchical manner until each subproblem is manageable. Thus, a book is decomposed into the task objects of chapters, the chapters into sections, the sections into paragraphs, and the paragraphs into sentences. Each sentence is approximately one unit of thought for both the author and the reader. Most designed objects have similar decompositions: computer programs, buildings, television sets, cities, paintings, and plays, for example. Some objects are more neatly and easily decomposed than others; some objects are easier to understand than others.

Similarly, task actions can be decomposed into smaller actions. A construction plan can be reduced to a series of steps; a baseball game has innings, outs, and pitches; and creating a business letter involves creating an address, date, addressee, body, signature, and so on.

In writing a business letter with a computer, the user has to integrate smoothly the three forms of knowledge. The user must have the high-level concept of writing (task action) a letter (task object), recognize that the letter will be stored as a file (computer object), and know the details of the `save` command (computer action and syntactic knowledge). The user must be fluent with the middle-level concept of composing a sentence and must recognize the mechanism for beginning, writing, and ending a sentence. Finally, the user must know the proper low-level details of spelling each word (task), comprehend the motion of the cursor on the screen (computer concept), and know which keys to press for each letter (syntactic knowledge).

Integrating the three forms of knowledge, the objects and actions, and the multiple levels of semantic knowledge is a substantial challenge that

takes great motivation and concentration. Learning 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 syntactic/semantic 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 9).

Designers of interactive systems can apply the syntactic/semantic model to systematize their efforts. Where possible, the semantics of the task objects should be made explicit and the user's task actions should be laid out clearly. Then the computer objects and actions can be identified, leaving the syntactic details for the end. In this way, designs appear to be more comprehensible to users and more independent of specific hardware.

2.3 PRINCIPLES: RECOGNIZE THE DIVERSITY

The remarkable diversity of human abilities, backgrounds, cognitive styles, and personalities challenges the interactive system designer. When multiplied by the wide range of situations, tasks, and frequencies of use, the set of 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 learning from a computer-assisted instruction lesson is a long way from a hotel reservations clerk serving customers for many hours a day.

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

2.3.1 Usage profiles

"Know the user" was the first principle in Hansen's (1971) list of user engineering principles. It's a simple idea, but a difficult goal and, unfortunately, an often undervalued goal. No one would argue against this principle, but many designers assume that they understand the users and their tasks. Successful designers are aware that other people learn, think, and solve problems in very different ways (see Section 1.5). Some users really do have an easier time with tables than graphs, with words instead of numbers, with slower rather than faster display rates, 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 electric power utility controllers.

All design should begin with an understanding of the intended users, including profiles of their age, sex, 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. 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 knowing the user 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, knowledgeable intermittent, and frequent users might lead to these differing design goals:

Novice users: This community is assumed to have no syntactic knowledge about using the system and probably little semantic knowledge of computer issues. They may even have shallow knowledge of the task and, worse still, they may arrive with anxiety about using computers that

inhibits learning. Overcoming these limitations is a serious challenge to the designer. Restricting vocabulary to a small number of familiar, consistently used terms is essential to begin developing the user's knowledge of the system. The number of possibilities should be kept small and the novice user should be able to carry out a few simple tasks to build confidence, reduce anxiety, and gain positive reinforcement from success. Informative feedback about the accomplishment of each task is helpful, and constructive, specific error messages should be provided when errors do occur. Carefully designed paper manuals and step-by-step online tutorials may be effective.

Knowledgeable intermittent users: Many people will be knowledgeable but intermittent users of a variety of systems. They will be able to maintain the semantic knowledge of the task and the computer concepts, but they will have difficulty maintaining the syntactic knowledge. The burden of memory will be lightened by simple and consistent structure in the command language, menus, terminology, and so on, and by the use of recognition rather than recall. Consistent sequences of actions, meaningful messages, and frequent prompts will all help to assure knowledgeable intermittent users that they are performing their tasks properly. Protection from danger is necessary to support relaxed exploration of features or attempts to invoke a partially forgotten command. These users will benefit from online help screens to fill in missing pieces of syntactic or computer semantic knowledge. Well-organized reference manuals will also be useful.

Frequent users: The knowledgeable "power" users are thoroughly familiar with the syntactic and semantic aspects of the system and seek to get their work done rapidly. They demand rapid response times, brief and less distracting 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. After gaining confidence from hands-on experience, the users can progress to ever greater levels of semantic concepts and the accompanying syntax. The learning plan should be governed by the progress through the task semantics. For users with strong knowledge of the task and computer semantics, rapid presentation of syntactic details 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. The progress is governed by the task domain, not by commands.

The level-structured approach must be carried out not only in the design of the software, but also in 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 prefer more densely packed displays than do novices. Finally, the pace of interaction may be varied from slow for novices to fast for frequent users.

2.3.2 Task profiles

After carefully drawing the user profile, the tasks must be identified. Task analysis has a long, but mixed history (Bailey, 1982). Every designer would agree that the set of tasks must be decided on 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 the 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 a set of commands, a menu tree, etc. 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:

1. Frequent actions might be performed by special keys, such as the four cursor arrows, the INSERT, and the DELETE key.
2. Intermediate frequency actions might be performed by a single letter plus the CTRL key, or by a selection from a pull-down menu. Examples include underscore, center, indent, subscript, or superscript.
3. Less frequent actions might require going to a command mode and typing the command name; for example, MOVE BLOCK or SPELLING CHECK.
4. Still less frequent actions or complex actions might require going through a sequence of menu selections or form fill-ins; for example, to change the printing format or to revise network protocol parameters.

A matrix of users and tasks can help sort out these issues (Figure 2.3). In each box, the designer can put a check mark to indicate that this user carries out this task. A more precise analysis would lead to inclusion of frequencies instead of simple check marks.

FREQUENCY OF TASK BY JOB TITLE

	Task				
	Query by Patient	Update Data	Query across Patients	Add Relations	Evaluate System
Job title					
Nurses	.14	.11			
Physicians	.06	.04			
Supervisors	.01	.01	.04		
Appointments personnel	.26				
Medical record maintainers	.07	.04	.04	.01	
Clinical researchers			.08		
Database programmers			.02	.02	.05

Figure 2.3: Hypothetical frequency of use data for a medical clinic information system. Queries by patient from appointments personnel are the highest frequency task.

2.3.3 Interaction styles

When the task analysis is complete and the semantics of the task objects and actions can be identified, the designer can choose from these primary interaction styles (Table 2.1):

- menu selection
- form fill-in
- command language
- natural language
- direct manipulation.

Chapters 3 through 5 explore these styles in detail, but first a comparative overview sets the stage.

INTERACTION STYLE	
ADVANTAGES	DISADVANTAGES
Menu selection	
shortens learning	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
easy to support error handling	
Form fill-in	
simplifies data entry	consumes screen space
requires modest training	
assistance is convenient	
permits use of form management tools	
Command language	
flexibilty	poor error handling
appeals to "power" users	requires substantial training and memorization
supports user initiative	
convenient for creating user defined macros	
Natural language	
relieves burden of learning syntax	requires clarification dialog
	may require more keystrokes
	may not show context
	unpredictable
Direct Manipulation	
visually presents task concepts	may be hard to program
easy to learn	may require graphics display and pointing devices
easy to retain	
errors can be avoided	
encourages exploration	
high subjective satisfaction	

Table 2.1: Advantages and disadvantages of the five primary interaction styles.

Menu selection: The users read a list of items, select the one most appropriate to their task, apply the syntax to indicate their selection, confirm the choice, initiate the action, and observe the effect. If the terminology and meaning of the items are understandable and distinct, then the users can accomplish their task with little learning or memorization and few keystrokes. The greatest benefit may be that there is a clear structure to decision making since only a few choices are presented at a 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 very 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. Dialog management tools to support menu selection are an enormous benefit in ensuring consistent screen design, validating completeness, and supporting maintenance.

Form fill-in: When data entry is required, menu selection usually becomes cumbersome, and form fill-in (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 fill-in interaction style, the 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, the 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 3 provides a thorough treatment of menus and form fill-in.

Command language: For frequent users, command languages provide a strong feeling of locus of control and initiative. The 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 computer concepts and syntax. Command languages and lengthier query or programming languages are the domain of the expert frequent users who often derive great satisfaction from mastering a complex set of semantics

and syntax. Chapter 4 covers command languages and natural language interaction in depth.

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.

Direct manipulation: When a clever designer can create a visual representation of the world of action, the users' tasks can be greatly simplified by allowing direct manipulation of the objects of interest. Examples include display editors, LOTUS 1–2–3, air traffic control systems, and video games. By pointing at visual representations of objects and actions, users can rapidly carry out tasks and immediately observe the results. Keyboard entry of commands or menu choices is replaced by cursor motion devices to select from a visible set of objects and actions. Direct manipulation is appealing to novices, easy to remember for intermittent users, and with careful design it can be rapid for frequent users. Chapter 5 describes direct manipulation and its applications.

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

2.4 EIGHT GOLDEN RULES OF DIALOG DESIGN

Later chapters cover constructive guidance for design of 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 include:

1. *Strive for consistency*. This principle is the most frequently violated one, and yet the easiest one to repair and avoid. Consistent sequences of actions should be required in similar situations, identical terminology should be used in prompts, menus, and help screens, and consistent commands should be employed throughout. Exceptions, such as nonprinting of passwords or no abbreviation 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 does the desire to reduce the number of interactions and increase the pace of interaction. Abbreviations, special keys, hidden commands, and macro facilities are appreciated by frequent knowledgeable users. Shorter response times and faster display rates are other attractions for frequent users.
3. *Offer informative feedback*. For every operator action there should be some system feedback. For frequent and minor actions the response can be very 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 explicitly showing changes (see direct manipulation in Chapter 5).
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 the operator the satisfaction of accomplishment, a sense of relief, the signal to drop contingency plans and options from his/her mind, and an indication that the way is clear to prepare for the next group of actions.
5. *Offer simple error handling*. As much as possible, design the system so the user cannot make a serious error. If an error is made, try to have the system detect the error and offer simple, comprehensible mechanisms for handling the error. The user should not have to retype the entire

command, but only need repair the faulty part. Erroneous commands should leave the system state unchanged or give instructions about restoring the system.

6. *Permit easy reversal of actions.* As much as possible, actions should be reversible. This relieves anxiety since the operator knows that errors can be undone, and encourages exploration of unfamiliar options. The units of reversibility may be a single action, a data entry, or a complete group of actions.
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, incapacity or difficulty in obtaining necessary information, and the inability to produce the action they want all build anxiety and dissatisfaction. Gaines (1981) captured part of this principle with his rule to "avoid acausality" and his encouragement to make users the initiators of actions rather than the responders.
8. *Reduce short-term memory load.* The limitation of human information processing in short-term memory ("seven plus or minus two chunks") requires that displays be kept simple, multiple page displays be consolidated, frequent window motion be reduced, and sufficient training time be permitted 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.

2.5 PREVENTING ERRORS

Users of text editors, database query facilities, air traffic control systems, and other interactive systems make mistakes far more frequently than might be expected. Ledgard et al. (1980) found that novice users of a 15-command subset of a text editor made mistakes in 19 percent of their commands. Experienced users made mistakes in 10 percent of their commands. In a more demanding environment, 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. Barber (1979) found that professional workers in a challenging decision-making job made errors in 7 percent to 46 percent of their transactions, depending on the response time of the computer system. Other studies are beginning to reveal the magnitude of the problem and the loss of productivity due to user errors.

One direction for reducing 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 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`.

But improved error messages 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 can be avoided by organizing screens and menus functionally, designing commands or menu choices to be distinctive, and making it difficult for users to do irreversible actions. Norman offers other guidelines such as "do not have

modes," offer feedback about the state of the system, and design for consistency of commands. Norman's analysis provides practical examples and a useful theory. The ensuing sections refine his analysis and describe three specific techniques for reducing errors by ensuring complete and correct actions: correct matching pairs, complete sequences, and correct commands.

2.5.1 Techniques for ensuring correct actions

Correct matching pairs. This is a common problem with many manifestations and several simple prevention strategies. Examples include 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 hopefully a more meaningful message such as UNMATCHED LEFT PARENTHESES.
- the " to close a string in BASIC. The command `10 PRINT "HELLO"` is in error if the rightmost " is missing.
- the @B or other markers to close a boldface, italic, or underscored text in word processors. If the text file contains `@BThis is boldface@B` then the three words between the @B markers appear in boldface on the printer. If the rightmost @B is missing, then the remainder of the file is printed in boldface.
- the termination of a centering command in a text formatter. Some text formatters have a pair of commands such as `.ON CENTER` and `.OFF CENTER` surrounding lines of text to be centered. The omission of the latter command causes the entire file to be centered.

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 using 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 and may prefer a milder form of protection. When the user types a left parenthesis, the screen displays a message in the lower left corner indicating the need for a right parenthesis, until it is typed.

Another approach is to replace the requirement for the ending marker. Many microcomputer versions of BASIC do not require an ending " to terminate a string. They use a carriage return to signal the closing of a string. Variants of this theme occur in line-oriented text editors that allow omission of the final / in a `CHANGE /OLD STRING/NEW STRING/` command. Many versions of LISP offer a special character, usually a right square bracket, to terminate all open parentheses.

In each of these cases, the designers have recognized a frequently occurring error and have found a way to eliminate the error situation.

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 turn signal lights on the left side of the car to flash. When a pilot lowers the landing gear, hundreds of steps and checks are invoked automatically.

This same concept can be applied to interactive uses of computers. For example:

- dialing up, setting communication parameters, logging on, and loading files is a frequently executed sequence for many users. Fortunately, most communications software

packages enable users to specify these processes once and then execute them by simply selecting the appropriate name.

- programming language loop constructs require a WHILE–DO–BEGIN–END or FOR–NEXT structure, but sometimes users forget to put the complete structure in or delete one component but not the other components. One solution would be for users to indicate that they wanted a loop, and the system could supply the complete and correct syntax, that would be filled in by the user. This approach reduces typing and the possibility of making a typographical error or a slip, such as the omission of one component. Conditional constructs require an IF–THEN–ELSE or CASE–OF–END structure; but again, users may forget a component when creating or deleting. Here again, if users could indicate that they wanted a conditional construct, the system could provide the syntactic template and prompt for the contents to be filled in (Teitelbaum & Reps, 1981).
- programming plans (Soloway et al., 1982) may contain several components that must be created and deleted one component at a time. The counter plan requires a data declaration of the integer variable, initialization to zero, incrementation, and a test. If the user could indicate that a counter plan is desired, then the system could provide the complete template, prompt the user for inclusion of each component, or merely remind the user of the need to complete the plan.
- a user of a text editor should be able to indicate that section titles are to be centered, in upper case, and underlined without having to issue a series of commands each time a section title is entered. Then if a change is made in style, for example, to eliminate underlining, a single command would guarantee that all commands were made correctly.
- air traffic controllers may formulate plans to change the altitude of a plane from 14,000 feet to 18,000 feet in two steps, but after raising the plane to 16,000 feet, the

controller may get distracted and 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 actually issued and the pattern of errors that people actually make.

Correct commands. Industrial designers recognize that successful products must be safe and must prevent the user from making incorrect use of the product. Airplane engines cannot be put into reverse until the landing gear have touched down, and cars cannot be put into reverse while traveling forward at faster than five miles per hour. Cameras prevent double exposures, even though this is sometimes desired, 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 computer systems: they invoke a command that is not available, make a menu selection choice that is not permitted, request a file that does not exist, or enter a data value that is 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 even for experts.

Whoever made the mistake and whatever were its causes, users must interrupt their planning to deal with the problem and their frustration in not getting what they wanted. As long as a command must be made up of a series of keystrokes on a keyboard, there is a substantial chance of making an error in entering the sequence of keypresses. Some keypressing sequences are more error-prone than others, especially those that require shifting or unfamiliar patterns. Reducing the number of keypresses can help, but it may place a greater burden on learning and memory since an entry with reduced keystrokes; for example, RM may be more difficult to remember than the full command name REMOVE (see Chapter 4).

Some systems offer automatic command completion that allows the users to type just a few letters of a meaningful command. The users 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 much to type for each command and must verify that the computer has made the completion that was intended.

Another approach is to have the computer offer the permissible commands, menu choices, or file names on the screen and let the user select with a pointing device, such as a mouse, lightpen, or arrow keys. This is effective if the screen has ample space, the display rate is rapid, and the pointing device is fast and accurate. When the list grows too long to fit on the available screen space, some approach to hierarchical decomposition must be used.

Imagine that the twenty commands of an operating system were constantly displayed on the screen. After selecting the PRINT command (or icon), the system automatically offers the list of thirty files for selection. Two lightpen, touchscreen, or mouse selections can be done in less time and with higher accuracy than can typing the command PRINT JAN–JUNE–EXPENSES.

In principle, a programmer need type a variable name only once. After it has been typed, the programmer can select it, thus eliminating the chance of a misspelling and an UNDECLARED VARIABLE message.

It is not always easy to convert a complex command into a small number of selections and reduce errors. Pointing devices are often crude, slow, and annoying to use. The Xerox Star and the Apple Macintosh are successful and practical applications of these concepts, but there is still room for further invention and application of this concept.

2.6 GUIDELINES: DATA DISPLAY

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. 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 (Lockheed, 1981; Gaines & Shaw, 1984; Rubinstein & Hersh, 1984; Brown, 1986).

2.6.1 Organizing the display

Smith and Mosier (1984) offer five high-level objectives for data display (page 93):

1. *Consistency of data display*. This principle is frequently violated, but it is easy to repair. During the design process, the terminology, abbreviations, formats, and so on should all be standardized and controlled by using 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 related to the tasks required to be performed with this data. This objective is served by rules for neat columns of data, left justification for alphanumeric data, right justification of integers, lining up decimal points, proper spacing, comprehensible labels, and appropriate use of coded values.

3. *Minimal memory load on user.* Do not require users to remember information from one screen for use on another screen. Arrange tasks such that completion occurs with few commands, minimizing the chance of forgetting to perform a step. Provide labels and common formats for novice or intermittent users.
4. *Compatibility of data display with data entry.* The format of displayed information should be clearly linked to the format of the data entry.
5. *Flexibility for user control of data display.* Users can get the information in the form most convenient for the task they are working on.

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 detailed comments for control room design come from a report from the Electric Power Research Institute (Lockheed, 1981):

- be consistent in labeling and graphic conventions
- standardize abbreviations
- use consistent format in all displays (headers, footers, paging, menus, etc.)
- present a page number on each display page and allow actions to call up a page by entering its page number
- present data only if they assist the operator
- present information graphically, where appropriate, 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 actually 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 operators in the development of new displays and procedures.

Chapter 8 further discusses data display issues.

2.6.2 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. Multiple techniques exist for attention getting:

1. Intensity: use two levels only.
2. Marking: underline, enclose in a box, point to with an arrow, or use an indicator such as an asterisk, bullet, dash, or an X.
3. Size: use up to four sizes.
4. Choice of fonts: use up to three fonts.
5. Inverse video: use normal or inverse.
6. Blinking: use blinking or nonblinking (2–4 hertz).
7. Color: use up to four standard colors, with additional colors reserved for occasional use.
8. Audio: use soft tones for regular positive feedback, harsh sounds for rare emergency conditions.

A few words of caution are necessary. There is a danger in creating cluttered displays by overuse of these techniques. Novices need simple, logically organized, and well-labeled displays that guide their actions. Expert operators 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 then it becomes more difficult to cluster items across color codes. Operator control over highlighting, for example, allowing the operator in an air traffic control environment to assign orange to aircraft above 18,000 feet, may provide a useful resolution to concerns about personal preferences. Highlighting can be accomplished by 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 rapidly alert operators, but a mechanism to suppress alarms must be provided. Testing is necessary to ensure that operators can distinguish among alarm levels. Prerecorded or synthesized messages are an intriguing alternative, but since they may interfere with communications among operators they should be used cautiously.

2.7 GUIDELINES: 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 (1984) offer five high-level objectives for data entry (page 19):

1. *Consistency of data entry transactions*. Similar sequences of actions under all conditions; similar delimiters, abbreviations, etc.
2. *Minimal input actions by user*. Fewer input actions mean greater operator productivity and usually less chance for error. Making a choice by a single keystroke, lightpen touch, finger press, etc., 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 the operators must move their hands from a keyboard to a separate input device, the advantage is defeated, because home row

position is lost. Experienced operators 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 an operator to enter the same information in two locations since it 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 operator, who still has the option of overriding by retyping.

3. *Minimal memory load on user.* Reduce the need for the operator 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 closely linked to the format of displayed information.
5. *Flexibility for user control of data entry.* Experienced operators may prefer to enter information in a sequence they can control. 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.

2.8 PROTOTYPING AND ACCEPTANCE TESTING

A critical component of clear thinking about interactive system design is the replacement of the vague and misleading notion of "user friendliness" with the five measurable quality criteria:

- time to learn
- speed of performance
- rate of errors by users

- subjective satisfaction
- retention over time.

Once the decision about the relative importance of each of the human factors quality criteria has been made, specific measurable objectives should be established to inform customers and users and to guide designers and implementers. The acceptance test plan for a system should be included in the requirements document and should be written before the design is made. Hardware and software test plans are regularly included in requirements documents; extending the principle to human interface development is natural (see Chapter 10).

The requirements document for a word processing system might include this acceptance test:

> The subjects will be 35 secretaries hired from an employment agency with no word processing experience, but typing skills in the 35 to 50 words per minute range. They will be given 45 minutes of training on the basic features. At least 30 of the 35 secretaries should be able to complete 80 percent of the typing and editing tasks in the enclosed benchmark test correctly within 30 minutes.

Another testable requirement for the same system might be:

> After four half days of regular use of the system, 25 out of these 35 secretaries should be able to carry out the advanced editing tasks in the second benchmark test within 20 minutes while making fewer than six errors.

This second acceptance test captures performance after regular use. The choice of the benchmark tests is critical and highly system dependent. The test materials and procedures must also be refined by pilot testing before use.

A third item in the acceptance test plan might focus on retention:

> After two weeks, at least 15 of the test subjects should be recalled and be required to perform the third benchmark test. In 40 minutes, at least 10 of the subjects must be able to complete 75 percent of the tasks correctly.

Such performance tests constitute the definition of "user friendly" for this system. By having an explicit definition, both the managers and the

designers will have a clearer understanding of the system goals and whether they have succeeded. The presence of a precise acceptance test plan will force greater attention to human factors issues during the design and ensure that pilot studies are run to determine if the project can meet the test plan goals.

In a programming workstation project, the early requirement for performance helped shape the nature of the interface. That requirement was:

> New professional programmer users should be able to sign on, create a short program, and execute it against a stored test data set, without assistance and within 10 minutes.

Specific goals in acceptance tests are useful, but competent test managers will notice and record anecdotal evidence, suggestions from participants, subjective reactions of displeasure or satisfaction, their own comments, and exceptional performance (both good and bad) by individuals. The precision of the acceptance test provides an environment in which unexpected events are most noticeable.

2.9 BALANCE OF AUTOMATION AND HUMAN CONTROL

The principles in the previous sections are in harmony with the goal of simplifying the user's task—eliminating human actions when no judgment is required. The 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. The computers should be used to keep track of and retrieve large volumes of data, follow preset patterns, and carry out complex mathematical or logical operations (Table 2.2 has 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

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
Remember principles and strategies	Make rapid and consistent responses to input signals
	Recall quantities of detailed information accurately
Retrieve pertinent details without a priori connection	Process quantitative data in prespecified ways
Draw upon experience & adapt decisions to situation	
Select alternatives if original approach fails	
Reason inductively: generalize from observations	
Act in unanticipated emergencies & novel situations	Perform repetitive preprogrammed actions reliably
	Exert great, highly controlled physical force
Apply principles to solve varied problems	
Make subjective evaluations	
Develop new solutions	
Concentrate on important tasks when overload occurs	Perform several activities simultaneously
	Maintain operations under heavy information load
Adapt physical response to changes in situation	Maintain performance over extended periods of time

Table 2.2. Relative capabilities of humans and machines. (Compiled from Brown, C. Marlin *Human-Computer Interface Design Guidelines*, New Jersey: Ablex Publishing Company, 1986; McCormick, E.J. *Human factors engineering*. New York: McGraw-Hill, 1970, pp. 20–21; and Estes, W.K. Is human memory obsolete? *American Scientist*, 1980, *68*, pp. 62–69.)

world is an "open system" (there are a nondenumerable number of unpredictable events and system failures). By contrast, computers constitute a "closed system" (there are only a denumerable number of predictable 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, avoid expensive failures, or increase product quality.

For example, in air traffic control, common operations include changes to altitude, heading, or speed. These are well understood and potentially automatable by a scheduling and route allocation algorithm, but the operators must be present to deal with the highly variable and unpredictable emergency situations. An automated system might successfully deal with high volumes of traffic, but what would happen if the airport manager changed runways because of turbulent weather? The controller would have to reroute planes quickly. But suppose one pilot called in to request special clearance to land because of a failed engine while a second pilot reported 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. The air traffic controller cannot just jump into the emergency; he or she must be intensely involved in the situation in order to make an informed, rapid, and optimal decision. In short, the real world situation is 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 in May, 1983. An Air Canada Boeing 727 jet had a fire on board, and the controller cleared away traffic and began to guide the plane in for a landing. The smoke was so bad that the pilot had trouble reading his instruments and then the onboard transponder burned out so that 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 the plane down quickly enough to save the lives of many, but not all, of the passengers.

The goal of system design in many applications is to give the operator sufficient information about current status and activities so that when intervention is necessary, the operator has the knowledge and the capacity to perform correctly. Increasingly, the human role will be to respond to such anomalies as unanticipated situations, failing equipment, improper human performance, and incomplete or erroneous data (Eason, 1980).

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