Exhibit 1014 – Part 2

coming 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, to reduce anxiety, and to 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. Users are attempting to relate their existing knowledge to the task objects and actions in the application, so distractions with computer concepts and the syntax are an extra burden.

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 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.

Expert frequent users: The expert "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 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, 1989). 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 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, 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.
- Less frequent actions might require going to a command mode and typing the command name—for example, `MOVE BLOCK` or `SPELLING CHECK`.
- 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 (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.

2.4.3 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: menu selection, form fillin, command language, natural language, direct manipulation (Table 2.1). Chapters 3 through 5

Frequency of Task by Job Title

Job title	*Query by patient*	*Update data*	*Query across patients*	*Add relations*	*Evaluate system*
Nurses	0.14	0.11			
Physicians	0.06	0.04			
Supervisors	0.01	0.01	0.04		
Appointments personnel	0.26				
Medical-record maintainers	0.07	0.04	0.04	0.01	
Clinical researchers			0.08		
Database programmers			0.02	0.02	0.05

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.

Table 2.1

Advantages and disadvantages of the five primary interaction styles.

Interaction Style	
Advantages	*Disadvantages*
menu selection	
shortens learning	imposes 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	
makes assistance convenient	
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	
is convenient for creating user-defined macros	
natural language	
relieves burden of learning syntax	requires clarification dialog
	may require more keystrokes
	may not show context
	is unpredictable
direct manipulation	
presents task concepts visually	may be hard to program
is easy to learn	may require graphics display and pointing devices
is easy to retain	
allows errors to be avoided	
encourages exploration	
permits high subjective satisfaction	

explore these styles in detail; here, we give a comparative overview to set the stage.

Menu selection In menu-selection systems, 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 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 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, 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, 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 fillin.

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 (discussed at the end of Chapter 4).

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 the objects of interest is possible. Examples of such systems 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 carry out tasks rapidly and observe the results immediately. 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, is easy to remember for intermittent users, and, with careful design, 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 fillin 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.

2.5 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, derived heuristically from experience, should be validated and refined:

1. *Strive for consistency.* This principle is the most frequently violated one, but the benefits of adherence are large. 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 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. 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 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 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 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 simple error handling.* As much as possible, design the system so the user cannot make a serious error. If an error is made, the system should detect the error and offer simple, comprehensible mechanisms for handling the error. The user should not have to retype the entire command, but rather should need to repair only the faulty part. Erroneous commands 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; it thus 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 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*.
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 Preventing 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 had some errors in almost half their spreadsheets. 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 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 can avoid 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.

2.6.1 Techniques for ensuring correct actions

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.

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, even better, a more meaningful message, such as `UNMATCHED LEFT PARENTHESES`.

Another error is failure to include the closing quotation mark (") to close a string in BASIC. The command `10 PRINT "HELLO"` is in error if the rightmost quotation mark is missing.

Similarly, a `@B` or other marker is required to indicate the end of 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.

A final example is omitting 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 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 and may prefer a milder form of protection. 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.

Another approach is to replace the requirement for the ending marker. Many microcomputer versions of BASIC do not require an ending quotation mark 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/` com-

mand. 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 (front and rear) 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, 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.

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 they delete one component but not the other components. One solution would be for users to indicate that they want a loop, and for the system to supply the complete and correct syntax, which 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.

Users of a text editor should be able to indicate that section titles are to be centered, set in upper-case letters, and underlined, without having to issue a series of commands each time they enter a section title. Then, if a change is made in style—for example, to eliminate underlining—a single command would guarantee that all section titles were 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 steps; after raising the plane to 16,000 feet, however, 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 5 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 computer systems: They invoke commands that are not available, type menu selection choices that are not permitted, 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.

Whoever made the mistake and whatever were its causes, users must interrupt their planning to deal with correcting the problem—and with 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 user to type just a few letters of a meaningful command. The user 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 key-

strokes 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.

Another approach is to have the computer offer the permissible commands, menu choices, or file names on the screen, and to let the user select 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. When the list grows too long to fit on the available screen space, some approach to hierarchical decomposition must be used.

Imagine that the 20 commands of an operating system were constantly displayed on the screen. After users select the `PRINT` command (or icon), the system automatically offers the list of 30 files for selection. Users can make two lightpen, touchscreen, or mouse selections in less time and with higher accuracy than they could by typing the command `PRINT JAN-JUNE-EXPENSES`.

In principle, a programmer needs to 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 thus to reduce errors. Pointing at long lists can be visually demanding and annoying if users are competent typists.

2.7 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 (see Chapters 8 and 13). 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 (Rubinstein and Hersh, 1984; Brown, 1988; Galitz, 1989). Inspirations for design guidelines can also be taken from graphics designers (Tufte, 1983, 1990).

2.7.1 Organizing the display

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

1. *Consistency of data display:* This principle is frequently violated, but violations are easy to repair. During the design process, the terminology,

abbreviations, formats, 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 use of coded values.
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 commands, 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.
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.

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, 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, 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.

2.7.2 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. Multiple techniques exist for attention getting:

Intensity: Use two levels only.

Marking: Underline, enclose in a box, point to with an arrow, or use an indicator such as an asterisk, bullet, dash, or an X.

Size: Use up to four sizes.

Choice of fonts: Use up to three fonts.

Inverse video: Use inverse coloring.

Blinking: Use blinking displays (2 to 4 hertz).

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).

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 overusing 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 this use makes it 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 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 operators rapidly, but a mechanism to suppress alarms must be provided. Testing is necessary to ensure that operators can distin-

guish 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: 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 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 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 operator, who still has the option of overriding by retyping.
3. *Minimal memory load on user:* When doing data entry, the operator 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 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 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
- Retention over time
- Subjective satisfaction

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 (Chapter 13).

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

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

Another testable requirement for the same system might be this:

> After 4 half-days of regular use of the system, 25 of these 35 secretaries should be able to carry out, within 20 minutes, the advanced editing tasks in the second benchmark test, and should make fewer than six errors.

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

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

> After 2 weeks, at least 15 of the test subjects should be recalled, and should perform the third benchmark test. In 40 minutes, at least 10 of the subjects should be able to complete 75 percent of the tasks correctly.

Such performance tests constitute the definition of "user friendly" for this system. By having an explicit definition, both the managers and the designers will gain 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 will ensure that pilot studies are run to determine whether the project can meet the test plan goals.

In a programming-workstation project, this early requirement for performance helped shape the nature of the interface:

> New professional programmers should be able to sign on, to create a short program, and to execute that program 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, such as 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.

2.10 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, to follow preset patterns, and to carry out complex mathematical or logical operations (Table 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

Table 2.2

Relative capabilities of humans and machines. (Compiled from Brown (1988); McCormick, E. J., *Human Factors Engineering*, McGraw-Hill, New York (1970), 20–21; and Estes, W. K., Is human memory obsolete? *American Scientist 68* (1980), 62–69.)

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

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 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 actions include changes to altitude, heading, or speed. These actions are well understood and are potentially 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 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. Air-traffic controllers cannot just jump suddenly into the emergency; they must be intensely involved in the situation 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 on an Air Canada Boeing 727. The jet had a fire on board, and the controller cleared other traffic from the flight paths and began to guide the plane in for a landing. The smoke was so bad 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 unexpected series of events.

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

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 operator.

2.11 Adaptive Agents and User Models versus Control Panels

The balance of automation and human control also emerges as an issue in systems for home and office automation. Many designers promote the notion of anthropomorphic agents that would wisely carry out the users's intents and anticipate needs (Norcio, 1989). These scenarios often show a responsive, butlerlike human being to represent the agent (a 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 Bob in Hewlett-Packard's 1990 video on computing in 1995). Others have described "knowbots," agents that traverse networks and scan large databases seeking information that the user might find interesting.

These fantasies are appealing; most people are attracted to the idea that a powerful functionary is continuously carrying out their tasks and watching out for their needs. The wish to create an 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. However, human–human interaction is not necessarily a good model for human–computer interaction. Many users have a strong desire to be in control and to have a sense of mastery over the system, so that they can derive feelings of accomplishment. Users usually seek predictable systems and shy away from complex unpredictable behavior. Simple task domain concepts should mask the underlying computational complexity, in the same way that turning on an automobile ignition is simple to the user 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 unchanged.

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. For example, several proposals suggest that, as users make menu selections more rapidly, indicating proficiency, advanced menu items or a command-line interface can be introduced. Automatic adaptations have been proposed for response time, length of messages, density of feedback, content of menus, order of menu items (see Section 3.3 for evidence against

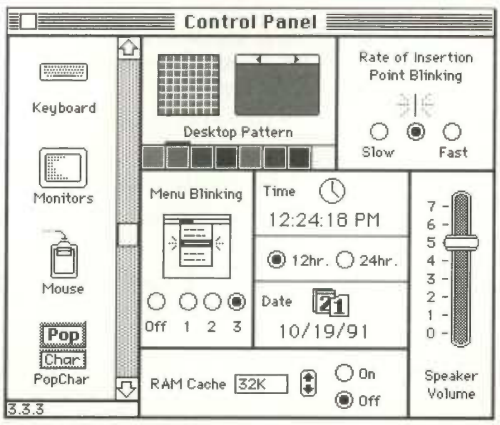
this strategy), type of feedback (graphic or tabular), and content of help screens. Advocates point to video games that increase the speed or number of dangers as users progress though stages of the game. However, games are quite different from most work situations, where users have external goals and motivations to accomplish their tasks. There is much discussion of user models, but little empirical evidence of their efficacy.

There are some opportunities to tailor system responses as a function of context, but unexpected behavior is a serious negative side effect that discourages use. If adaptive systems make surprising changes, users must pause to see what has changed. Then, they may become anxious because they may not be able to predict the next change, to interpret what has happened, or to restore the system to the previous state. The agent metaphor and "active, adaptive, intelligent" systems seem to be more attractive to designers who believe that they are creating something lifelike and even magical, than they are to users who may feel anxious and unable to control the system.

An alternative to agents and user models may be to expand the control-panel metaphor. 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, date formats, placement and format of menus, or color schemes (Figure 2.4 and 2.5). Some software packages allow users to set parameters such as the speed in games or the usage level as in HyperCard (from browsing to editing buttons 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

Figure 2.4

Control panel from the Apple Macintosh with scrolling list of specific controls. (Copyright Apple Computer, Inc., Cupertino, CA. Used with permission.)



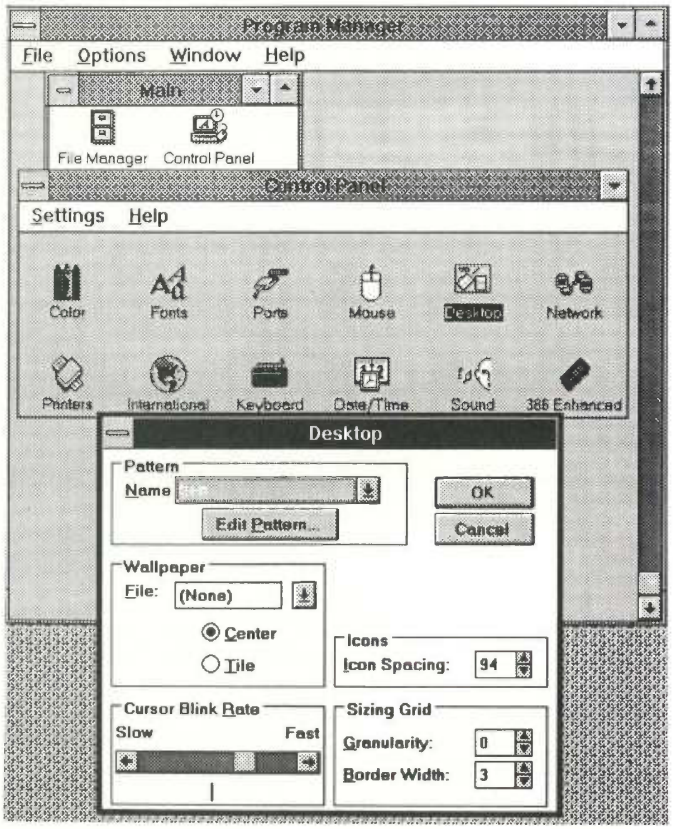


Figure 2.5

Control panel from Microsoft Windows graphical environment version 3.0. (Screen shot ©1985–1991 Microsoft Corporation. Reprinted with permission from Microsoft Corporation, Redmond, WA.)

sheets of word processors, specification boxes of query facilities, and scheduling software that carries out processes at regular intervals or when triggered by other processes.

Computer control panels, like cruise-control mechanisms in automobiles, are designed to convey the sense of control that users seem to expect. Increasingly, complex processes are specified by direct-manipulation programming (see Chapter 5) or by dialog-box specifications in graphical user interfaces. An effective design enables users to have a comprehensible task domain model of what the system does, and to make multiple choices rapidly.