ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 2, Issue 2, pp: (415-426), Month: April-June 2014, Available at: www.researchpublish.com

USER INTERFACE DESIGN

S.Sridevi

Department of Computer Science & Engineering, Saveetha School Of Engineering

Saveetha University, Chennai

Abstract: **The user interface is arguably the most important element of a computer-based system or product. If the interface is poorly designed, the user's ability to tap the computational power of an application may be severely hindered. In fact, a weak interface may cause an otherwise well-designed and solidly implemented application to fail. Three important principles guide the design of effective user interfaces: (1) place the user in control, (2) reduce the user's memory load, and (3) make the interface consistent. To achieve an interface that abides by these principles, an organized design process must be conducted. User interface design begins with the identification of user, task, and environmental requirements. Task analysis is a design activity that defines user tasks and actions using either an elaborative or object-oriented approach. Once tasks have been identified, user scenarios are created and analyzed to define a set of interface objects and actions. This provides a basis for the creation of screen layout that depicts graphical design and placement of icons, definition of descriptive screen text, specification and titling for windows, and specification of major and minor menu items. Design issues such as response time, command and action structure, error handling, and help facilities are considered as the design model is refined. A variety of implementation tools are used to build a prototype for evaluation by the user. The user interface is the window into the software. In many cases, the interface molds a user's perception of the quality of the system. If the "window" is smudged, wavy, or broken, the user may reject an otherwise powerful computer-based system.**

Keywords: **The Golden Rules, User Interface Design, Task Analysis and Modeling, Interface Design Activities, Implementation Tools, Design Evaluation.**

I. INTRODUCTION

The blueprint for a house (its architectural design) is not complete without a representation of doors, windows, and utility connections for water, electricity, and telephone (not to mention cable TV). The "doors, windows, and utility connections" for computer software make up the interface design of a system.

Interface design focuses on three areas of concern: (1) the design of interfaces between software components, (2) the design of interfaces between the software and other nonhuman producers and consumers of information (i.e., other external entities), and (3) the design of the interface between a human (i.e., the user) and the computer.

On user interface design, Ben Shneiderman [SHN90] states:

The problems to which Shneiderman alludes are real. It is true that graphical user interfaces, windows, icons, and mouse picks have eliminated many of the most horrific interface problems. But even in a "Windows world," we all have encountered user interfaces that are difficult to learn, difficult to use, confusing, unforgiving, and in many cases, totally frustrating. Yet, someone spent time and energy building each of these interfaces, and it is not likely that the builder created these problems purposely.

User interface design has as much to do with the study of people as it does with technology issues. Who is the user? How does the user learn to interact with a new computer-based system? How does the user interpret information produced by the system? What will the user expect of the system? These are only a few of the many questions that must be asked and answered as part of user interface design.

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 2, Issue 2, pp: (415-426), Month: April-June 2014, Available at: www.researchpublish.com

II. THE GOLDEN RULES

On interface design, Theo Mandel [MAN97] coins three "golden rules":

1. Place the user in control.

2. Reduce the user's memory load.

3. Make the interface consistent.

These golden rules actually form the basis for a set of user interface design principles that guide this important software design activity.

Place the User in Control

During a requirements-gathering session for a major new information system, a key user was asked about the attributes of the window-oriented graphical interface. "What I really would like," said the user solemnly, "is a system that reads my mind. It knows what I want to do before I need to do it and makes it very easy for me to get it done. That's all, just that."

My first reaction was to shake of my head and smile, but I paused for a moment. There was absolutely nothing wrong with the user's request. She wanted a system that reacted to her needs and helped her get things done. She wanted to control the computer, not have the computer control her. Most interface constraints and restrictions that are imposed by a designer are intended to simplify the mode of interaction. But for whom? In many cases, the designer might introduce constraints and limitations to simplify the implementation of the interface. The result may be an interface that is easy to build, but frustrating to use.

Mandel [MAN97] defines a number of design principles that allow the user to maintain control:

Define interaction modes in a way that does not force a user into unnecessary or undesired actions. An interaction mode is the current state of the interface. For example, if *spell check* is selected in a word-processor menu, the software moves to a spell checking mode. There is no reason to force the user to remain in spell checking mode if the user desires to make a small text edit along the way. The user should be able to enter and exit the mode with little or no effort.

Provide for flexible interaction. Because different users have different interaction preferences, choices should be provided. For example, software might allow a user to interact via keyboard commands, mouse movement, a digitizer pen, or voice recognition commands. But every action is not amenable to every interaction mechanism. Consider, for example, the difficulty of using keyboard command (or voice input) to draw a complex shape.

Allow user interaction to be interruptible and undoable. Even when involved in a sequence of actions, the user should be able to interrupt the sequence to do something else (without losing the work that had been done). The user should also be able to "undo" any action.

Streamline interaction as skill levels advance and allow the interaction to be customized. Users often find that they perform the same sequence of interactions repeatedly. It is worthwhile to design a "macro" mechanism that enables an advanced user to customize the interface to facilitate interaction.

Hide technical internals from the casual user. The user interface should move the user into the virtual world of the application. The user should not be aware of the operating system, file management functions, or other arcane computing technology.

In essence, the interface should never require that the user interact at a level that is "inside" the machine (e.g., a user should never be required to type operating system commands from within application software).

Design for direct interaction with objects that appear on the screen. The user feels a sense of control when able to manipulate the objects that are necessary to perform a task in a manner similar to what would occur if the object were a physical thing. For example, an application interface that allows a user to "stretch" an object (scale it in size) is an implementation of direct manipulation.

Reduce the User's Memory Load

The more a user has to remember, the more error-prone will be the interaction with the system will be. It is for this reason that a well-designed user interface does not tax the user's memory. Whenever possible, the system should "remember"

pertinent information and assist the user with an interaction scenario that assists recall. Mandel[MAN97] defines design principles that enable an interface to reduce the user's memory load:

Reduce demand on short-term memory. When users are involved in complex tasks, the demand on short-term memory can be significant. The interface should be designed to reduce the requirement to remember past actions and results. This can be accomplished by providing visual cues that enable a user to recognize past actions, rather than having to recall them.

Establish meaningful defaults. The initial set of defaults should make sense for the average user, but a user should be able to specify individual preferences. However, a "reset" option should be available, enabling the redefinition of original default values.

Define shortcuts that are intuitive. When mnemonics are used to accomplish a system function (e.g., alt-P to invoke the print function), the mnemonic should be tied to the action in a way that is easy to remember (e.g., first letter of the task to be invoked).

The visual layout of the interface should be based on a real world metaphor. For example, a bill payment system should use a check book and check register metaphor to guide the user through the bill paying process. This enables the user to rely on well-understood visual cues, rather than memorizing an arcane interaction sequence.

Disclose information in a progressive fashion. The interface should be organized hierarchically. That is, information about a task, an object, or some behavior should be presented first at a high level of abstraction. More detail should be presented after the user indicates interest with a mouse pick. An example, common to many word-processing applications, is the underlining function. The function itself is one of a number of functions under a text style menu. However, every underlining capability is not listed. The user must pick underlining, then all underlining options (e.g., single underline, double underline, dashed underline) are presented.

Make the Interface Consistent

The interface should present and acquire information in a consistent fashion. This implies that (1) all visual information is organized according to a design standard that is maintained throughout all screen displays, (2) input mechanisms are constrained to a limited set that are used consistently throughout the application, and (3) mechanisms for navigating from task to task are consistently defined and implemented. Mandel [MAN97] defines a set of design principles that help make the interface consistent:

Allow the user to put the current task into a meaningful context. Many interfaces implement complex layers of interactions with dozens of screen images. It is important to provide indicators (e.g., window titles, graphical icons, consistent color coding) that enable the user to know the context of the work at hand. In addition, the user should be able to determine where he has come from and what alternatives exist for a transition to a new task.

Maintain consistency across a family of applications. A set of applications(or products) should all implement the same design rules so that consistency is maintained for all interaction.

If past interactive models have created user expectations, do not make changes unless there is a compelling reason to do so. Once a particular interactive sequence has become a de facto standard (e.g., the use of alt-S to save a file), the user expects this in every application he encounters. A change (e.g., using alt-S to invoke scaling) will cause confusion. The interface design principles discussed in this and the preceding sections provide basic guidance for a software engineer. In the sections that follow, we examine the interface design process itself.

III. USER INTERFACE DESIGN

The overall process for designing a user interface begins with the creation of different models of system function (as perceived from the outside). The human- and computer-oriented tasks that are required to achieve system function are then delineated; design issues that apply to all interface designs are considered; tools are used to prototype and ultimately implement the design model; and the result is evaluated for quality.

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 2, Issue 2, pp: (415-426), Month: April-June 2014, Available at: www.researchpublish.com

Interface Design Models

Four different models come into play when a user interface is to be designed. The software engineer creates a *design model*, a human engineer (or the software engineer) establishes a *user model*, the end-user develops a mental image that is often called the *user's model* or the *system perception*, and the implementers of the system create a *system image* [RUB88]. Unfortunately, each of these models may differ significantly. The role of interface designer is to reconcile these differences and derive a consistent representation of the interface.

A design model of the entire system incorporates data, architectural, interface, and procedural representations of the software. The requirements specification may establish certain constraints that help to define the user of the system, but the interface design is often only incidental to the design model. 1 The user model establishes the profile of end-users of the system. To build an effective user interface, "all design should begin with an understanding of the intended users, including profiles of their age, sex, physical abilities, education, cultural or ethnic background, motivation, goals and personality" [SHN90]. In addition, users can be categorized as

• **Novices.** No *syntactic knowledge*2 of the system and little *semantic knowledge*3 of the application or computer usage in general.

• **Knowledgeable, intermittent users.** Reasonable semantic knowledge of the application but relatively low recall of syntactic information necessary to use the interface.

• **Knowledgeable, frequent users.** Good semantic and syntactic knowledge that often leads to the "power-user syndrome"; that is, individuals who look for shortcuts and abbreviated modes of interaction.

The system perception (user's model) is the image of the system that end-users carry in their heads. For example, if the user of a particular word processor were asked to describe its operation, the system perception would guide the response. The accuracy of the description will depend upon the user's profile (e.g., novices would provide a sketchy response at best) and overall familiarity with software in the application domain. A user who understands word processors fully but has worked with the specific word processor only once might actually be able to provide a more complete description of its function than the novice who has spent weeks trying to learn the system.

The system image combines the outward manifestation of the computer-based system (the look and feel of the interface), coupled with all supporting information (books, manuals, videotapes, help files) that describe system syntax and semantics.

When the system image and the system perception are coincident, users generally feel comfortable with the software and use it effectively. To accomplish this "melding" of the models, the design model must have been developed to accommodate the information contained in the user model, and the system image must accurately reflect syntactic and semantic information about the interface.

The models described in this section are "abstractions of what the user is doing or thinks he is doing or what somebody else thinks he ought to be doing when he uses an interactive system"[MON84].

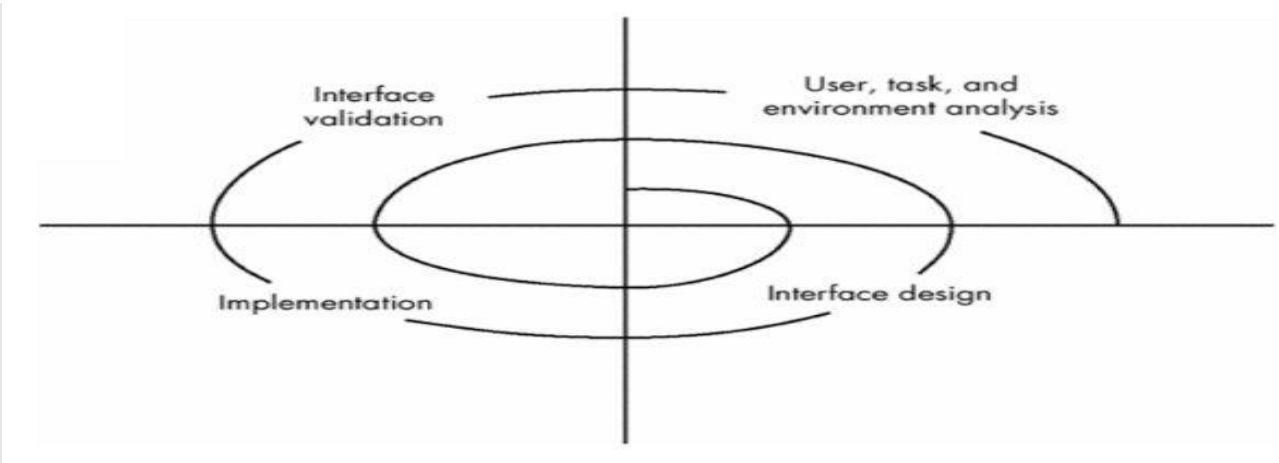

FIGURE1. **User Interface Design Process (Spiral Model)**

In essence, these models enable the interface designer to satisfy a key element of the most important principle of user interface design: "Know the user, know the tasks."

The User Interface Design Process

The design process for user interfaces is iterative and can be represented using a spiral model. Referring to Figure 1, the user interface design process encompasses four distinct framework activities [MAN97]:

1. User, task, and environment analysis and modeling

2. Interface design

3. Interface construction

4. Interface validation

The spiral shown in Figure 1 implies that each of these tasks will occur more than once, with each pass around the spiral representing additional elaboration of requirements and the resultant design. In most cases, the implementation activity involves prototyping—the only practical way to validate what has been designed.

The initial analysis activity focuses on the profile of the users who will interact with the system. Skill level, business understanding, and general receptiveness to the new system are recorded; and different user categories are defined. For each user category, requirements are elicited. In essence, the software engineer attempts to understand the system perception for each class of users.

Once general requirements have been defined, a more detailed task analysis is conducted. Those tasks that the user performs to accomplish the goals of the system are identified, described, and elaborated (over a number of iterative passes through the spiral).

The analysis of the user environment focuses on the physical work environment. Among the questions to be asked are

• Where will the interface be located physically?

• Will the user be sitting, standing, or performing other tasks unrelated to the interface?

• Does the interface hardware accommodate space, light, or noise constraints?

• Are there special human factors considerations driven by environmental factors?

The information gathered as part of the analysis activity is used to create an analysis model for the interface. Using this model as a basis, the design activity commences.

The goal of interface design is to define a set of interface objects and actions (and their screen representations) that enable a user to perform all defined tasks in a manner that meets every usability goal defined for the system.

The implementation activity normally begins with the creation of a prototype that enables usage scenarios to be evaluated. As the iterative design process continues, a user interface tool kit may be used to complete the construction of the interface.

Validation focuses on (1) the ability of the interface to implement every user task correctly, to accommodate all task variations, and to achieve all general user requirements;(2) the degree to which the interface is easy to use and easy to learn; and (3) the users' acceptance of the interface as a useful tool in their work. Therefore, there is no need to attempt to specify every detail (for the analysis or design model) on the first pass. Subsequent passes through the process elaborate task detail, design information, and the operational features of the interface.

IV. TASK ANALYSIS AND MODELING

Task analysis for interface design uses either an elaborative or object-oriented approach but applies this approach to human activities.

Task analysis can be applied in two ways. As we have already noted, an interactive, computer-based system is often used to replace a manual or semi-manual activity. To understand the tasks that must be performed to accomplish the goal of the

activity, a human engineer4 must understand the tasks that humans currently perform (when using a manual approach) and then map these into a similar (but not necessarily identical) set of tasks that are implemented in the context of the user interface. Alternatively, the human engineer can study an existing specification for a computer-based solution and derive a set of user tasks that will accommodate the user model, the design model, and the system perception.

Regardless of the overall approach to task analysis, a human engineer must first define and classify tasks. We have already noted that one approach is stepwise elaboration. For example, assume that a small software company wants to build a computer-aided design system explicitly for interior designers. By observing an interior designer at work, the engineer notices that interior design comprises a number of major activities: furniture layout, fabric and material selection, wall and window coverings selection, presentation (to the customer), costing, and shopping. Each of these major tasks can be elaborated into subtasks. For example, furniture layout can be refined into the following tasks: (1) draw a floor plan based on room dimensions;(2) place windows and doors at appropriate locations; (3) use furniture templates to draw scaled furniture outlines on floor plan; (4) move furniture outlines to get best placement; (5) label all furniture outlines; (6) draw dimensions to show location; (7) draw perspective view for customer. A similar approach could be used for each of the other major tasks.

Subtasks 1–7 can each be refined further. Subtasks 1–6 will be performed by manipulating information and performing actions within the user interface. On the other hand, subtask 7 can be performed automatically in software and will result in little direct user interaction. The design model of the interface should accommodate each of these tasks in a way that is consistent with the user model (the profile of a "typical" interior designer) and system perception (what the interior designer expects from an automated system).

An alternative approach to task analysis takes an object-oriented point of view. The human engineer observes the physical objects that are used by the interior designer and the actions that are applied to each object. For example, the furniture template would be an object in this approach to task analysis. The interior designer would *select* the appropriate template, *move* it to a position on the floor plan, *trace* the furniture outline and so forth. The design model for the interface would not provide a literal implementation for each of these actions, but it would define user tasks that accomplish the end result (drawing furniture outlines on the floor plan).

V. INTERFACE DESIGN ACTIVITIES

Once task analysis has been completed, all tasks (or objects and actions) required by the end-user have been identified in detail and the interface design activity commences. The first interface design steps [NOR86] can be accomplished using the following approach:

1. Establish the goals5 and intentions for each task.

2. Map each goal and intention to a sequence of specific actions.

3. Specify the action sequence of tasks and subtasks, also called a *user scenario,* as it will be executed at the interface level.

4. Indicate the state of the system; that is, what does the interface look like at the time that a user scenario is performed?

5. Define control mechanisms; that is, the objects and actions available to the user to alter the system state.

6. Show how control mechanisms affect the state of the system.

7. Indicate how the user interprets the state of the system from information provided through the interface.

Always following the golden rules discussed above, the interface designer must also consider how the interface will be implemented, the environment (e.g., display technology, operating system, development tools) that will be used, and other elements of the application that “sit behind” the interface.

Defining Interface Objects and Actions

An important step in interface design is the definition of interface objects and the actions that are applied to them. That is, a description of a user scenario is written. Nouns (objects) and verbs (actions) are isolated to create a list of objects and actions.

Once the objects and actions have been defined and elaborated iteratively, they are categorized by type. Target, source, and application objects are identified. A *source object* (e.g., a report icon) is dragged and dropped onto a *target object* (e.g., a printer icon). The implication of this action is to create a hard-copy report. An *application object* represents application-specific data that is not directly manipulated as part of screen interaction. For example, a mailing list is used to store names for a mailing. The list itself might be sorted, merged, or purged (menu-based actions) but it is not dragged and dropped via user interaction.

When the designer is satisfied that all important objects and actions have been defined (for one design iteration), *screen layout* is performed. Like other interface design activities, screen layout is an interactive process in which graphical design and placement of icons, definition of descriptive screen text, specification and titling for windows, and definition of major and minor menu items is conducted. If a real world metaphor is appropriate for the application, it is specified at this time and the layout is organized in a manner that complements the metaphor.

To provide a brief illustration of the design steps noted previously, we consider a user scenario for an advanced version of the *Safe Home* system. In the advanced version, *Safe Home* can be accessed via modem or through the Internet. A PC application allows the homeowner to check the status of the house from a remote location, reset the *Safe Home* configuration, arm and disarm the system, and (using an extra cost video option6) monitor rooms within the house visually. A preliminary user scenario for the interface follows:

Scenario: The homeowner wishes to gain access to the *Safe Home* system installed in his house. Using software operating on a remote PC (e.g., a notebook computer carried by the homeowner while at work or traveling), the homeowner determines the status of the alarm system, arms or disarms the system, reconfigures security zones, and views different rooms within the house via preinstalled video cameras. To access *Safe Home* from a remote location, the homeowner provides an identifier and a password. These define levels of access (e.g., all users may not be able to reconfigure the system) and provide security. Once validated, the user (with full access privileges) checks the status of the system and changes status by arming or disarming *Safe Home.* The user reconfigures the system by displaying a floor plan of the house, viewing each of the security sensors, displaying each currently configured zone, and modifying zones as required. The user views the interior of the house via strategically placed video cameras. The user can pan and zoom each camera to provide different views of the interior.

Home owner tasks:

• *accesses* the *SafeHome* system

• *enters* an **ID** and **password** to allow remote access

• *checks* **system status**

• *arms* or *disarms SafeHome* system

• *displays* **floor plan** and **sensor locations**

• *displays* **zones** on floor plan

• *changes* **zones** on floor plan

• *displays* **video camera locations** on floor plan

• *selects* **video camera** for viewing

• *views* **video images** (4 frames per second)

• *pans* or *zooms* the **video camera**

Objects (boldface) and actions (italics) are extracted from this list of homeowner tasks. The majority of objects noted are application objects. However, video camera location (a source object) is dragged and dropped onto video camera (a target object) to create a video image (a window with video display).

A preliminary sketch of the screen layout for video monitoring is created (Figure2). To invoke the video image, a **video camera location** icon, *C*, located in **floor plan** displayed in the **monitoring window** is selected. In this case a camera location in the living room, LR, is then dragged and dropped onto the **video camera** icon in the upper left-hand portion of

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 2, Issue 2, pp: (415-426), Month: April-June 2014, Available at: www.researchpublish.com

the screen. The **video image window** appears, displaying streaming video from the camera located in the living room (LR). The **zoom** and **pan** control slides are used to control the magnification and direction of the video image. To select a view from another camera, the user simply drags and drops a different **camera location** icon into the**camera** icon in the upper left-hand corner of the screen.

The layout sketch shown would have to be supplemented with an expansion of each menu item within the menu bar, indicating what actions are available for the

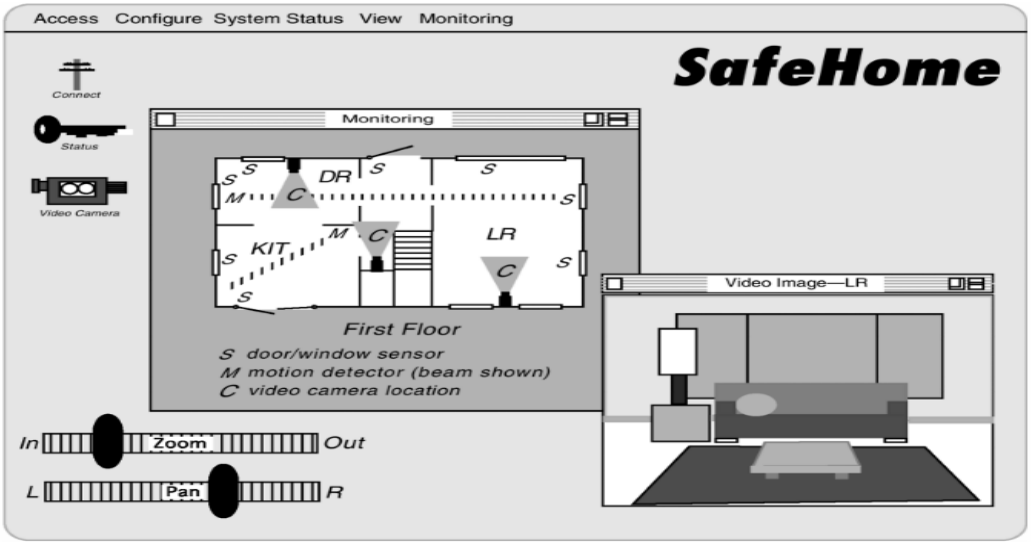

FIGURE 3.

Example – *SafeHome* (Preliminary Screen Layout)

Video monitoring mode (state). A complete set of sketches for each homeowner task noted in the user scenario would be created during the interface design.

Design Issues

As the design of a user interface evolves, four common design issuesalmost always surface: system response time, user help facilities, error information handling, and command labeling. Unfortunately, many designers do not address these issues until relatively late in the design process (sometimes the first inkling of a problem doesn't occur until an operational prototype is available). Unnecessary iteration, project delays, and customerfrustration often result. It is far better to establish each as a design issue to be considered at the beginning of software design, when changes are easy and costs are low.

System response time is the primary complaint for many interactive applications. In general, system response time is measured from the point at which the user performs some control action (e.g., hits the return key or clicks a mouse) until the software responds with desired output or action.

System response time has two important characteristics: length and variability. If the*length* of system response is too long, user frustration and stress is the inevitable result. However, a very brief response time can also be detrimental if the user is being paced by the interface. A rapid response may force the user to rush and therefore make mistakes.

Variability refers to the deviation from average response time, and in many ways, it is the most important response time characteristic. Low variability enables the user to establish an interaction rhythm, even if response time is relatively long. For example, a 1-second response to a command is preferable to a response that varies from 0.1 to 2.5 seconds. The user is always off balance, always wondering whether something "different" has occurred behind the scenes.

Almost every user of an interactive, computer-based system requires help now and then. In some cases, a simple question addressed to a knowledgeable colleague can do the trick. In others, detailed research in a multivolume set of "user

manuals" may be the only option. In many cases, however, modern software provides on-line help facilities that enable a user to get a question answered or resolve a problem without leaving the interface.

Two different types of help facilities are encountered: integrated and add-on[RUB88]. An *integrated help facility* is designed into the software from the beginning. It is often context sensitive, enabling the user to select from those topics that are relevant to the actions currently being performed. Obviously, this reduces the time required for the user to obtain help and increases the "friendliness" of the interface.

An *add-on help facility* is added to the software after the system has been built. In many ways, it is really an on-line user's manual with limited query capability. The user may have to search through a list of hundreds of topics to find appropriate guidance, often making many false starts and receiving much irrelevant information. There is little doubt that the integrated help facility is preferable to the add-on approach. A number of design issues [RUB88] must be addressed when a help facility is considered:

• Will help be available for all system functions and at all times during system interaction? Options include help for only a subset of all functions and actions or help for all functions.

• How will the user request help? Options include a help menu, a special function key, or a HELP command.

• How will help be represented? Options include a separate window, a reference to a printed document(less than ideal), or a one- or two-line suggestion produced in a fixed screen location.

• How will the user return to normal interaction? Options include a return button displayed on the screen, a function key, or control sequence.

• How will help information be structured? Options include a "flat" structure in which all information is accessed through a keyword, a layered hierarchy of information that provides increasing detail as the user proceeds into the structure, or the use of hypertext.

Error messages and warnings are "bad news" delivered to users of interactive systems when something has gone awry. At their worst, error messages and warnings impart useless or misleading information and serve only to increase user frustration. There are few computer users who have not encountered an error of the form:

VI. SEVERE SYSTEM FAILURE -- 14A

Somewhere, an explanation for error 14A must exist; otherwise, why would the designers have added the identification? Yet, the error message provides no real indication of what is wrong or where to look to get additional information. An error message presented in this manner does nothing to assuage user anxiety or to help correct the problem.

In general, every error message or warning produced by an interactive system should have the following characteristics:

• The message should describe the problem in jargon that the user can understand.

• The message should provide constructive advice for recovering from the error.

•The message should indicate any negative consequences of the error (e.g., potentially corrupted data files) so that the user can check to ensure that they have not occurred (or correct them if they have).

• The message should be accompanied by an audible or visual cue. That is, a beep might be generated to accompany the display of the message, or the message might flash momentarily or be displayed in a color that is easily recognizable as the "error color."

• The message should be "nonjudgmental." That is, the wording should never place blame on the user. Because no one really likes bad news, few users will like an error message no matter how well designed. But an effective error message philosophy can do much to improve the quality of an interactive system and will significantly reduce user frustration when problems do occur.

The typed command was once the most common mode of interaction between user and system software and was commonly used for applications of every type. Today, the use of window-oriented, point and pick interfaces has reduced

reliance on typed commands, but many power-users continue to prefer a command-oriented mode of interaction. A number of design issues arise when typed commands are provided as a mode of interaction:

• Will every menu option have a corresponding command?

• What form will commands take? Options include a control sequence (e.g., alt-P), function keys, or a typed word.

• How difficult will it be to learn and remember the commands? What can be done if a command is forgotten?

• Can commands be customized or abbreviated by the user?

As we noted earlier, conventions for command usage should be established across all applications. It is confusing and often error-prone for a user to type alt-D when a graphics object is to be duplicated in one application and alt-D when a graphics object is to be deleted in another. The potential for error is obvious.

VII. IMPLEMENTATION TOOLS

Once a design model is created, it is implemented as a prototype,7 examined by users (who fit the user model described earlier) and modified based on their comments. To accommodate this iterative design approach, a broad class of interface design and prototyping tools has evolved. Called *user-interface toolkits* or *user-interface developmentsystems* (UIDS), these tools provide components or objects that facilitate creation of windows, menus, device interaction, error messages, commands, and many other elements of an interactive environment.

Using prepackaged software components to create a user interface, a UIDS provides built-in mechanisms [MYE89] for

• managing input devices (such as a mouse or keyboard)

• validating user input

• handling errors and displaying error messages

• providing feedback (e.g., automatic input echo)

• providing help and prompts

• handling windows and fields, scrolling within windows

• establishing connections between application software and the interface

• insulating the application from interface management functions

• allowing the user to customize the interface

These functions can be implemented using either a language-based or graphical approach.

VIII. DESIGN EVALUATION

Once an operational user interface prototype has been created, it must be evaluated to determine whether it meets the needs of the user. Evaluation can span a formality spectrum that ranges from an informal "test drive," in which a user provides impromptu feedback to a formally designed study that uses statistical methods for the evaluation of questionnaires completed by a population of end-users.

The user interface evaluation cycle takes the form shown in Figure 3. After the design model has been completed, a first-level prototype is created. The prototype is evaluated by the user, who provides the designer with direct comments about the efficacy of the interface. In addition, if formal evaluation techniques are used (e.g., questionnaires, rating sheets), the designer may extract information from these data (e.g., 80 percent of all users did not like the mechanism for saving data files).

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 2, Issue 2, pp: (415-426), Month: April-June 2014, Available at: www.researchpublish.com

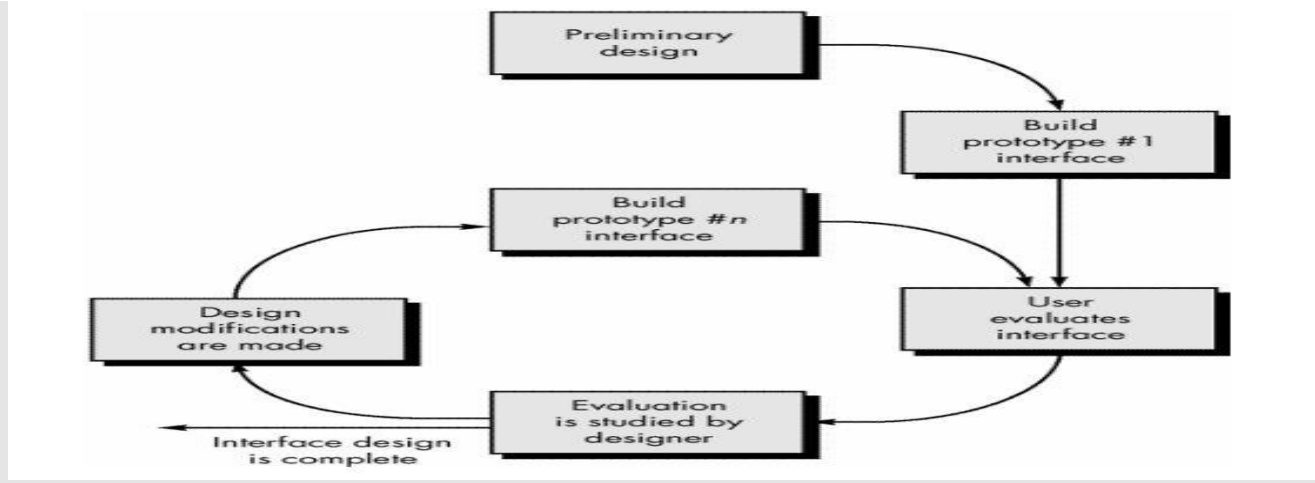

FIGURE 2. User Interface Evaluation Cycle

Design modifications are made based on user input and the next level prototype is created. The evaluation cycle continues until no further modifications to the interface design are necessary.

The prototyping approach is effective, but is it possible to evaluate the quality of a user interface before a prototype is built? If potential problems can be uncovered and corrected early, the number of loops through the evaluation cycle will be reduced and development time will shorten. If a design model of the interface has been created, a number of evaluation criteria [MOR81] can be applied during early design reviews:

1. The length and complexity of the written specification of the system and its interface provide an indication of the amount of learning required by users of the system.

2. The number of user tasks specified and the average number of actions per task provide an indication of interaction time and the overall efficiency of the system.

3. The number of actions, tasks, and system states indicated by the design model imply the memory load on users of the system.

4. Interface style, help facilities, and error handling protocol provide a general indication of the complexity of the interface and the degree to which it will be accepted by the user.

Once the first prototype is built, the designer can collect a variety of qualitative and quantitative data that will assist in evaluating the interface. To collect qualitative data, questionnaires can be distributed to users of the prototype. Questions can be all (1) simple yes/no response, (2) numeric response, (3) scaled (subjective) response, or (4) percentage (subjective) response. Examples are

1. Were the icons self-explanatory? If not, which icons were unclear?

2. Were the actions easy to remember and to invoke?

3. How many different actions did you use?

4. How easy was it to learn basic system operations (scale 1 to 5)?

5. Compared to other interfaces you've used, how would this rate—top 1%, top10%, top 25%, top 50%, bottom 50%?

If quantitative data are desired, a form of time study analysis can be conducted. Users are observed during interaction, and data—such as number of tasks correctly completed over a standard time period, frequency of actions, sequence of actions, time spent "looking" at the display, number and types of errors, error recovery time, time spent using help, and number of help references per standard time period—are collected and used as a guide for interface modification.

REFERENCES

[1] [LEA88] Lea, M., "Evaluating User Interface Designs," *User Interface Design for Computer Systems*, Halstead Press (Wiley), 1988.

[2] [MAN97] Mandel, T., *The Elements of User Interface Design*, Wiley, 1997.

[3] [MON84] Monk, A. (ed.), *Fundamentals of Human-Computer Interaction*, Academic Press, 1984.

[4] [MOR81] Moran, T.P., "The Command Language Grammar: A Representation for the User Interface of Interactive Computer Systems," *Intl. Journal of Man-Machine Studies*, vol. 15, pp. 3–50.

[5] [MYE89] Myers, B.A., "User Interface Tools: Introduction and Survey, *IEEE Software*,January 1989, pp. 15–23.

[6] [NOR86] Norman, D.A., "Cognitive Engineering," in *User Centered Systems Design*, Lawrence Earlbaum Associates, 1986.

[7] [RUB88] Rubin, T., *User Interface Design for Computer Systems*, Halstead Press (Wiley), 1988.

[8] [SHN90] Shneiderman, B., *Designing the User Interface*, 3rd ed., Addison-Wesley, 1990.

[9] Pressman, Roger S.,Software engineering: a practitioner's approach / Roger S. Pressman.—5th ed.,2001.