INTERNATIONAL BUSINESS SCHOOLS COMPUTING QUARTERLY

A Publication of the International Business Schools Computing Association

Volume 5, Number 1

Spring 1993

On the design of intuitive user interfaces

Hossein Saiedian
Scott Ames
University of Nebraska, Omaha

Abstract

The design of user-interfaces is a complicated process because many factors have to be considered, and many of these factors are usually difficult to overcome. The most difficult factors to overcome are those that are directly related to the users' behavior. Users differ from one other and come from different background and experience. To overcome these factors, interface designers attempt to build interfaces that are intuitive. Such interfaces are expected to be easily learned by a variety of users. In this paper we discuss important issues related to the design of intuitive user interfaces.

Introduction

User interface design has become increasingly important as computer use grows. In fact, it has become a distinct design activity during the development of many modern applications. Its goal is to establish the layout and interaction mechanisms through which a user interacts with the computer. If the interface is easy to use, intuitive and forgiving, the user will be inclined to make good use of the application. Since during the human-computer interaction, the visual, tactile, and auditory senses of humans predominate, the design of user interface design should encompass both the study of users (e.g., who they are) as well as technological issues. According to Fischer (1989), recent developments in the area of interface design include:

· workstations with bitmapped displays and pointing devices that provide a new technological base on which today's applications can be built,[1]

· innovative applications that have drawn attention to the computer's interaction capabilities, and

· the complexity of today's software that has made better communication techniques a necessity.

This paper looks at the possibility of designing intuitive user interfaces. The user interface can be defined as everythingthat the user interacts with when operating the computer. This includes software, hardware, task procedures, documentation, and training materials. We must admit that this is a very broad definition. A narrower definition would emphasize the mechanisms through which a dialogue between software and humans is realized. This paper thus focuses on software related issues. Issues of good and bad interface design techniques are discussed. Some existing interfaces are examined for those properties that help to make an interface intuitive and those properties that make an interface not so intuitive. In addition, we discuss the importance of maintaining consistency in the human-computer interface and review the interface design process and the importance of prototyping. Usability factors are discussed at the end.

Design Factors

There are many factors that limit how intuitive a user interface can be. The most important and difficult factor to overcome is that people are different. What may be intuitive for one person may be completely foreign to another.[2] In some cases, previous computer experience will not play a role in how difficult the system is to learn while in other cases the amount of previous computer experience that a person has does affect whether or not that person will find a user interface intuitive. Someone with years of computer experience may find almost all interfaces intuitive. Someone new to computers may find almost all interfaces foreign. For these reasons, it is very difficult if not impossible to design a user interface that will be intuitive to everyone. Thus human skills, personality, cognitive style and background may all play an important role. For example, a user's skill level will have a significant impact on the way she/he responds to tasks that are generated because of interaction with the computer, while a user's personality (e.g., risk taking vs. risk avoidance) or cognitive style (e.g., reflective vs. impulsive) may demand a different accommodation.

Even though it might be impossible to design a user interface that is intuitive to everyone, there are still some things that the designer can do to help make a user interface intuitive for some users. There are also some things that designers can do to make user interfaces nonintuitive for almost everyone. Rosson, Maass, and Kellogg (1988) have given certain principles of user interface design based on questions given to designers. These principles are shown in Table 1.

Table 1: Principles of Interface Design

- Input/Output
  - minimize input movements
  - maximize input channels
  - visually attractive
  - careful use of language
  - screen and key consistency
  - highly visual interfaces
- Dialog
  - process continuity
  - modeless or mode feedback
  - use of menus, prompts
  - context-dependent messages
  - user control
  - natural response time
- Conceptual Model
  - use of analogy, metaphor
  - match user expectations
  - start with user model
  - make examples concrete
  - minimize semantic primitives
  - novice-expert path
- General
  - easy-to-use
  - personalize
  - make it simple
  - follow standards
  - user friendly
  - understand user needs

Table 2: Characteristics of Consistent User Interface

- Physical
  - screens look the same
  - function keys to the same thing
- Dialog
  - coherent dialog style, e.g. windows, menus
  - same procedure works the same way
  - same command has the same meaning
- Conceptual Model
  - simulate the world
  - use the same function across applications
  - use generic actions

Conceptual Models and the Role of Metaphors

Of the principles listed in Table 1, those listed under the heading *Conceptual Model* are the most important for creating intuitive user interfaces. Conceptual modeling includes establishing a user model. The user model must include those profiles of the intended users that will affect the interface design as well as user's perception of the system, i.e., the image of the system that the user carries in his or her mind. When the users' perception and the actual user interface coincide, the users will feel comfortable with the software and will use it effectively and efficiently. Thus the objective of conceptual modeling is to create a user model of the interface from the users' perception.

One popular way to establish an effective conceptual model is to use *metaphors*. A metaphor is a way of representing objects in a computer system that users can equate with concepts with which they are already familiar with. The idea behind using a metaphor is that if the computer system is similar to the original system on which the metaphor is based, it will be easier for users to learn how to use the computer system. In fact the power of metaphors has been explored in other areas of information system. For example, Mason (1991) examines the role of metaphors in strategic information systems.

We have observed that one of the best examples of a metaphor is the desktop metaphor used by many computers such as the Macintosh. The desktop metaphor is a paper-based model of a persons desk. Since most people have had some experience with handling, writing, and manipulating paper-based information, this model seems appropriate for a user interface. Furthermore, we have noticed that one problem with this is that if the paperless office ever comes, the desktop metaphor will be obsolete. We have discovered that this is not just a potential problem for the desktop metaphor. All metaphors are frozen in time and could possibly become obsolete. We believe this can be attributed to the fact that metaphors are built to fit one situation. When that situation changes, the metaphor needs to be replaced or modified to fit the new situation.

Problems also arise when trying to pick a metaphor for a user interface. We have observed that the more complex the computer system, the harder it will be to find a good metaphor. To be intuitive, the metaphor needs to be simple enough so that even beginners can understand and use the system. If the metaphor is complex and needs to be explained to be used, it is not intuitive. A metaphor based interface may not even be appropriate for the application, so why spend valuable design time trying to find one.

Another problem with using metaphors arises when the metaphor is extended to include functionality not present in the original system upon which the metaphor was based. The perfect example of this is the trash can icon used in the Macintosh interface. To eject a diskette, the user can throw its icon in the trash. Rather than deleting the contents of the disk as the trash can icon implies, the disk is ejected. This is completely counter-intuitive. We have seen looks of terror on people's faces when we have done this to their diskette.

Consistency in Interface

Another feature of design that helps to make a user interface intuitive is consistency. To design a consistent interface, consistency must first be defined. Grudin (1989) notes that in a 2-day workshop in 1988, 15 experts could not define consistency. Even if a definition for consistency cannot be found, most people have an idea of what it means and can tell the difference between a consistent interface and one that is not consistent. Consistency implies that the interface commands, menus, and windows should have similar formats. If an interface is command-driven, then all parameters to commands should be passed in a similar way, and the punctuation should be similar. For example, if a command accepts both flags and filenames (as in UNIX), then the way flags and filenames are distinguished should be similar in all commands. Furthermore, if a character such as *k* is used as *keep* flag in one command, it should not be used as *kill* flag in another command. The same concept can be applied to menu-driven interfaces. If the *Escape* key is used for exiting from one menu, it should be used for the same purpose in all other menus. The objectives in maintaining a consistent interface is that when users learn about one command or menu of the interface, that knowledge will be applicable to all other commands and menus in the system. Rosson, Maass, and Kellogg (1988) provide a number of characteristics for the design of consistent interfaces. These characteristics are shown in Table 2.

Grudin (1989) identifies three types of consistency. Internal consistency is when the interface to one part of the software system is consistent with the interfaces of all the other parts of the software system. This has to do with such things as physical layout, command naming, and selection techniques. External consistency to other applications is when the interface of one application is consistent with the interfaces to other applications on the same computer system. This is very important if an interface is to be intuitive. Since most people use only one computer system, it is easier to learn a new application if the new application is consistent with other applications that the person already knows how to use. External consistency to the real world is when the metaphor used by the system is consistent with features found in the real world. This is also important if the user interface is to be intuitive.

Gould, Boies, and Lewis (1991) suggest that one way in which designers can attempt to create consistent interfaces is by creating interface design standards. Standards help to

separate the style of the application from the content of the application. According to Potosnak (1988), the problem with standards is that they can only define a very low level of consistency describing details of the user interface. This low level of consistency does not make the system any easier to use if the conceptual model of the task is wrong. For example, if it takes 17 pop-up menus to get to the most frequently used function, consistent design of the pop-up menus doesn't make the system easy to use. If the system is not easy to use, it is impossible for it to be intuitive. Furthermore, while it is easy to set standards on paper, it is usually hard to meet standards in design.

The User-Interface Design Process

Generally speaking, the software development life cycle consists of four major phases. These phases include *requirements analysis*, *software design*, *implementation*, followed by *testing*. During the requirements analysis, the software scope is established, its requirements are identified, models of information flow, and operational behavior are created. During the software design, the designers produce a more detailed representation of the software to be built later. Implementation involves translating the design representations into code to be understood by the computer while testing is the ultimate review and validation of the software specifications, design, and coding.

User interface design, as its title implies, is a design activity and is dealt with during the design phase. However, since a computer system always has a human element and since it is the interface that defines the human computer interaction,[3] it is important to analyze and specify the expected human-computer interaction early during the analysis. That is, the expected functions of the software under development have to be examined in the context of required interaction with the human and the precise meaning of each function is to be defined. Furthermore, other elements such as hardware, database, operating systems, etc. that are combined with the software to form the user's environment must be studied to plan the most effective user interface design.

The design phase itself has traditionally been divided into *preliminary* and *detailed* design sub-phases. However, because of the increasing importance of human factors and human-computer interactions, the design phase described by most researchers includes a distinct set of activities known as user interface design during which important human factors are taken into consideration to design the most effective interface.

The human-computer interaction part of design is variable and complex. Traditional tools, methods, and techniques do not work as well. New technology adds new input devices, styles, and techniques, but programming languages do not change to include these new technologies. Because of this, new design methods that include human factor issues must be adopted. While the problem with this is that it adds to the design cost, the benefits are that it enhances the product and helps to ensure its acceptance.

Gould, Boies, and Clayton (1991) offer four design activities that will help designers create user interfaces. First, early focus on the users is essential. Designers should have direct contact with the users. This helps to ensure that the user interface will be acceptable to the users. Second, the design of the user interface must be integrated with the design of the rest of the system. All aspects of the design should take place in parallel. Third, early and continual user testing should take place. This provides feedback to the designers about the interface. Fourth, the design process should be iterative. The designers should be able to go back to previous design phases and modify the system based on the user's feedback.

Gould, Boies, and Clayton further contend that following these steps will lead to systems that are easy to learn, contain the right functions, and are well liked. This goes a long way in helping to make a user interface intuitive. By continually getting feedback from the users of the system, designers can redo the user interface until the users of the system find it intuitive. This method does have problems because iterative design is risky. Designers generally don't want to have to go back and redesign parts of the system. Another problem is that interface design has not benefitted as much from modularization and code reuse as other aspects of design have. In typical software systems, one half of any new code that is added to a new version of the system is related to the user interface.

The need for continual testing has led researchers to consider using prototyping techniques for designing the user interface. Prototyping techniques allow an iterative user-interface design as follows:

- Create a user-interface design
- Implement using prototyping tools
- Allow users to evaluate the design
- Modify the design based on the users comments until no more changes are requested.

To accommodate the above, a variety of tools known as *user-interface development system (UIDS)* have been developed. These tools provide facilities for creating menus, windows, management of input devices, scrolling mechanisms, and so forth. Some tools (such as the X-window toolkit) even allow the users to customize their interface (e.g., enlarging or reducing the size of windows, changing the color of back- or foreground, choosing font style, etc.). The prototyping cycle takes the form shown in Figure 1. The first-level prototype is created and shown to the users; the users evaluate the efficacy of the interface and provide comments to the designer. Interface design modifications are made to accommodate user's requests and the next-level prototype is cre-

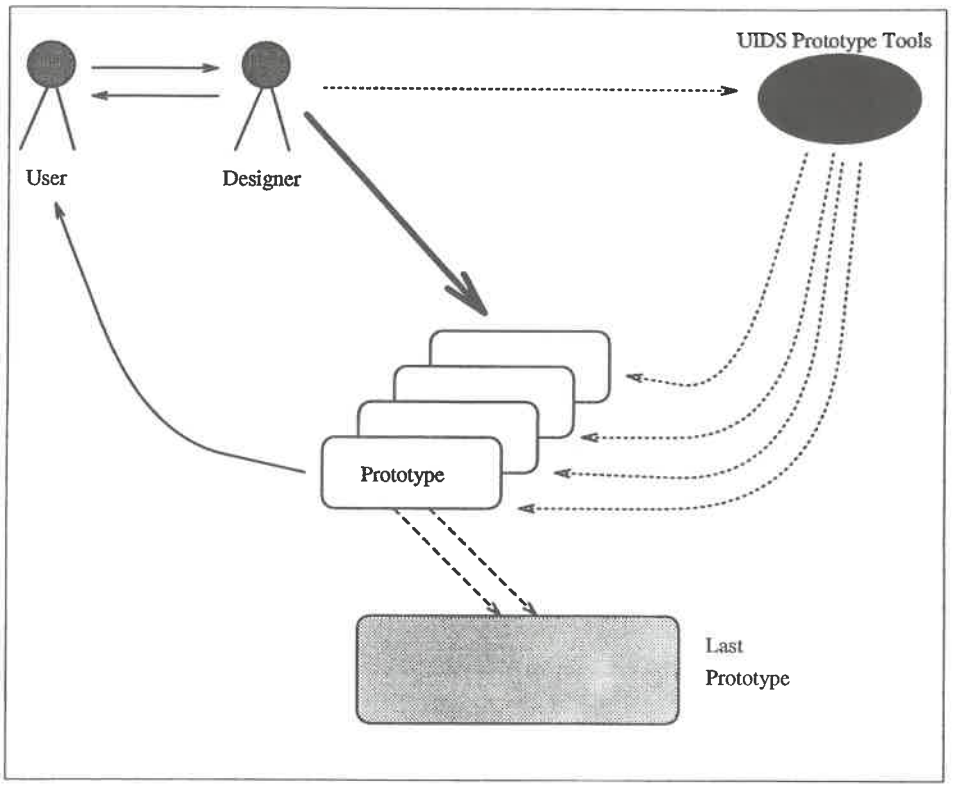

Figure 1: User-Interface Prototype Cycle

```
type shape
  superclasses: ()
  subclasses: (triangle, square)
  actions: (createShape, deleteShape, rotateShape)
  attributes: (position, angle, color)

triangle, square
  superclasses: (shape)
  subclasses: ()
  actions: (createShape, deleteShape, rotateShape)
  attributes: (position, angle, color)

color: set(1,1) of (white, gray, black)
angle: range[0..360] of integer
position: range[x:(0..maxX),y:(0..maxY)] of integer
```

Figure 2: Typical Features of UIDE

Table 3: Intuitive Interface Design Features

- simple and natural dialogue
- minimize the user's memory load
- provide feedback
- provide shortcuts
- error prevention
- speak the user's language
- be consistent
- provide clearly marked exits
- provide good error messages

ated. The cycle continues until no further changes are requested by the users.

User Interface Management Systems

Fischer (1989) suggests that the key to helping designers build better user interfaces is through the use of interface design tools that provide exploratory programming environments and rapid prototyping methods. According to Sutcliffe and Wang (1991), the ideal situation would be to have intelligent CASE tools to aid the designer. Gould, Boies, and Lewis (1991) discuss one type of design tool called a *User Interface Management System (UIMS)*. A UIMS helps a software designer design the user interface for the software. The advantage of using a UIMS is that it becomes possible to separate the user interface code from the rest of the code. This reduces the risks involved with making changes to the interface. A UIMS also promotes the reuse of code. One problem is that most UIMS work is still in the research stage. Another problem is that no one is sure whether or not a UIMS can produce today's advanced user interfaces. Until it becomes clear whether or not a UIMS can be used to produce today's advanced user interfaces, a UIMS is not going to be sufficient to produce an intuitive user interface.

An advanced UIMS reported by Foley, Kim, and Kovacevic (1989) is called *User Interface Design Environment (UIDE)*. It uses a knowledge-based representation of the interface's conceptual design based on an object oriented data model and an operation oriented control model. UIDE allows a class hierarchy of objects with single inheritance. Objects can have properties and actions that can be performed on the objects. Actions can have information associated with them that are required by the action. Actions can also have preconditions and postconditions.

This particular system works at a higher level of abstraction than other UIMS. The focus is moved away from things such as command names, screen design, and menu organization. This allows the designer to focus more on the conceptual model of the interface. A good conceptual model is essential if a user interface is to be intuitive.

Figure 2 illustrates some of the features of UIDE. Once definitions such as these are created, a series of transformations are applied to create the user interface.

Usability

For a user interface to be intuitive, it must be usable or easy to use. Molich and Nielson (1990) offer usability principles that are shown in Table 3.
Two of these features, feedback and good error messages, are essential if a user interface is to be intuitive. Although one should not be needed if the user interface is completely intuitive, a good help system is also required to make a system usable.

Sutcliffe and Wang (1991) suggest that the system should provide user feedback messages at various points during its operation. Feedback should be given after input is accepted to acknowledge the input. Guidance should be given before input is expected explaining what input is required. Error messages should be given when the user makes a mistake. Status information should be given at the start of task sequences and at subtask boundaries to inform the user about what is happening.

Good error messages are very important if the interface is to be intuitive. According to Nielson (1990), error messages should restate the user's input as it was interpreted by the system. This allows the user to figure out what went wrong. Nielson also observes that error messages should be specific as to what is wrong. An example of an error message that violates this property comes from the Macintosh computer system. To execute an application, the user is allowed to "double-click" on the icon of a document that was created by that particular application. The application is supposed to execute and automatically load the document that was selected. This usually works, but sometimes there is an error message that says the application is busy or missing. The problem with this is that the solution to the problem depends on whether or not the application is busy or if the application is missing. If the application is busy, it is not a very serious error. However, if the application is missing, the error is more serious. The error message should specify whether or not the application is busy or missing so that the user can take the appropriate actions. If the user does not know what action to take, then the user interface is not intuitive.

A good help system is also required if a computer system is to be intuitive. Carroll and Aaronson (1988) suggest that one way in which this can be done is to provide contextual help. This is also referred to as context dependent or context sensitive help. This type of help system provides help to the user based on what the user is currently doing. For example, if the system is asking the user for the page length, the user should be able to get help on what the computer is asking for at that particular moment.

Prager, Lamberti, and Gardner (1990) describe a system called *Real-time Explanation and SuggestiON* (REASON). REASON is an intelligent user-assistant prototype for a windowed, multitasking environment.[4] It uses an inference engine to solve problems that arise from the user's activity. If the user makes a mistake, REASON can offer dynamically generated suggestions. The user also has the ability to query REASON in a natural language. Systems such as this can be very helpful for the beginning user who is trying to learn a new system. However, as an experiment by Carroll and Aaronson (1988) shows, it can backfire.

Carroll and Aaronson carried out an experiment that simulated an intelligent help system. Users thought that they were using a system with intelligent help when it was really a human that was generating the suggestions and error messages that the users were seeing. The messages were of two types: *how-it-works* and h*ow-to-do-it*. How-it-works messages described goal based objectives without mentioning specific procedures. How-to-do-it messages described the actual procedures that should be followed by users to do something. The experiment revealed several interesting things. First, people are creative at generating errors and misconceptions. This makes it very difficult for a computer program to intelligently generate error messages. Second, the line between being annoying and actually helping the users is very fine. After the experiment, many users said that many of the messages would appear at inappropriate times. This also makes it difficult for a computer program to intelligently generate error messages. It remains to be seen whether or not an intelligent user assistant can make a user interface intuitive.

Human Usability Factors

There are many human factors that determine whether or not a particular interface will be considered usable by its users. As stated earlier, skill level of the users may play an important part in the user-interface. In fact, usability depends heavily on the skill level of the users. Creating an interface that is usable by all users is very difficult. Kantorowitz and Sudarsky (1989) claim that proficient users will generally find a command line interface more usable than a menu driven or icon-based interface. On the other hand, beginning users may find a menu driven or icon-based interface easier to use.

To solve the problem of different user skill levels, many systems provide both command line interfaces and menu driven interfaces. The problem with these systems is that users must complete a task and then switch to a different interface style. Kantorowitz and Sudarsky (1989) describe an *Adaptable User Interface* (AUI). It is capable of switching dialogue modes in the middle of a command. For example, a user could start in a command line interface mode. Then, if the user forgot some of the parameters, they could switch to a menu driven mode where they could select the rest of the parameters from a menu. This allows a beginning user to use a completely menu driven user interface. A more experienced user could use the command line user interface combined with the menu interface until they learned all of the parameters for the commands that they regularly use. An experienced user could use only the command line user interface without being bothered by the menus. A user interface such as this is quite a bit more intuitive than a system with only one fixed dialogue mode.

Different people like to operate in different ways. To create usable interfaces, user interface designers need to consider this when designing interfaces. Habermann (1991) notes that many current user interfaces use the concepts of windows, menus, buttons, and mice. He claims that many people do not think in these terms. For example, architects work with sketch pads, not windows. Architects also do not select a pencil from a menu, they just pick it up and use it. The problem is that the concepts employed by architects is different from the concepts provided by today's interfaces. Eventually, users will be able to work with an interface that is designed on their own terms. Virtual reality systems will make it possible for an architect to pick up a virtual pencil and draw. The architect will then be able to walk through their creation. Because of their ability to let the user work within a customized environment designed in terms of the user, virtual reality systems are one of the most promising new technologies that will help to make user interfaces intuitive.

To summarize, when planning an interface that would adapt to the capabilities and limitations of people, the designers must consider the sociological context in which the software will be used (e.g., the users skills and background) as well as the psychological aspects (e.g., brightness, loudness) and physical variables (e.g., intensity). The types of interaction dialogue that would best suit the users need must be considered. The varieties include form-filling, function-key based, menu-driven, natural language, command language, or graphical interaction. Of importance is the level of help that the interface provides, such as instructive input prompts, informative error messages, and useful status information and feedback. An excellent discussion of related topics regarding human-computer interaction and the design of computer information system is given in (Garlach & Kuo 1991).

Designing a Usable Interface

With so many factors involved in whether or not a user interface is usable, designers need to take many things into account when designing user interfaces. Habermann (1991) identifies three approaches for making user interfaces easier to use. First, the designer can analyze current interfaces to find places for improvement. Second, the designer can create new interaction techniques using existing technology. Third, the designer can create new technology for interaction. This last approach is where things such as virtual reality and multimedia fit in. Fischer (1989) offers the following points that will help designers create usable user interfaces:

- take advantage of technology
- use user interface construction kits
- provide exploratory environments for users
- use prototypes for testing by users
- promote natural communication
- help the user make intelligent choices

A question that arises during the design of a user interface is what type of interface should be designed. Picking an appropriate interface has a lot to do with whether or not a user finds an interface easy to use and, therefore, more intuitive. Many people feel that an icon-based interface such as the interface used by the Macintosh is very easy to use. However, according to Potosnak (1989), research shows that the interface style doesn't determine ease of use. The way in which the interface is designed is more important in determining ease of use. Potosnak also describes an experiment performed by John Whiteside and others at Digital Equipment Corporation with seven different systems. Two were icon-based, one was menu driven, and four were command line driven. 76 users participated in the experiment. The users were divided into three different types. The first type contained new users with very little computer experience. The second type contained transfer users. Transfer users are defined as people that use computers every day, but had never used any of the seven systems before. The third type contained system users. System users are defined as people that use computers every day, and had used many of the seven systems before. The experiment showed that new users performed better on the command and menu driven interfaces than on the icon-based interface. Transfer users also performed better on the command and menu driven interfaces. System users performed well on all three types of interfaces.

One technique to help insure that an interface is usable is for designers to perform a usability *walkthrough*. A usability walkthrough is a systematic review of a user interface design on paper. Three types of people do the walkthrough: representatives for the expected users, the product developers, and human factor professionals. These people are given a task to perform and hard copies of screens. They are then asked to describe how they would perform the task. The main advantage of doing a usability walkthrough is that the data collected can be used during the design of the interface. Also, if the users don't think that the interface is very intuitive, the entire interface can be thrown out and a new one designed. This is possible because at the point that the usability walkthrough is done, little money has been spend on the design of the interface. A potential problem is that not everything can be simulated on paper.

Conclusion

User interface design is a complicated process. There are many factors that the designer needs to consider when creating user interfaces. Many of these factors are difficult to overcome. The most difficult factors to overcome are those that are human related. People will always be different from one another. They will always have different backgrounds and experiences. Designing interfaces that take into account even a small number of the possible variables dealing with differences in people adds great complexity to the design process. For this reason alone, it is very difficult to create an interface that will be intuitive for every user.

Metaphors and consistency are two more important factors in user interface design. These two factors can help a user interface be intuitive. However, if done incorrectly these two factors can make an interface nonintuitive. Unfortunately, it is very easy to do these incorrectly.

A final very important factor is usability. If the user interface is not usable, it will not be intuitive. There are so many factors that determine whether or not a user interface is usable that it is impossible for a designer to consider all of them. Unfortunately, if an interface is to be intuitive, all usability factors should be considered.

Taken as a whole, all of these factors make it is very difficult, if not impossible, using current techniques and technology, to create a user interface that is intuitive. As newer, more sophisticated software engineering techniques and tools become available and as new technologies such as virtual reality systems and multimedia become available, future user interfaces may be able to advertise that they are truly intuitive.

References

[1]Carroll, J. and Aaronson, A. "Learning by Doing with Simulated Intelligent Help" *Communications of the ACM*, September 1988.

[2]Fischer, G. "Human-Computer Interaction Software: Lessons Learned, Challenges Ahead" *IEEE Software*, January 1989.

[3]Foley, J., Kim, C., Kovacevic, S. "Defining Interfaces at a High Level of Abstraction" *IEEE Software*, January 1989.

[4]Garlach, J and Kuo, F. "Understanding Human-Computer Interactions for Information System Design," *MIS Quarterly*, 15:4, December 1991.

[5]Gould, John D., Boies, Stephen J., and Lewis, Clayton "Making Usable, Useful, Productivity Enhancing Computer Applications" *Communications of the ACM*, January 1991.

[6]Grudin, J. "The Case Against User Interface Consistency" *Communications of the ACM*, October 1989.

[7]Habermann, F. "Giving Real Meaning to 'easy-to-use' Interfaces" *IEEE Software*, July 1991.

[8]Kantorowitz, E. and Sudarsky, O. "The Adaptable User Interface" *Communications of the ACM*, November 1989.

[9]Lamberti, D. and Wallace, D. "Intelligent Interface De-

sign," *MIS Quarterly*, 14:3, September 1990.

[10]Mason, R. "The Role of Metaphors in Strategic Information Systems," *Journal of Management Information Systems*, 8:2, Fall 1991.

[11]Molich, R. and Nielson, J. "Improving a Human-Computer Dialogue" *Communications of the ACM*, March 1990.

[12]Nielson, J. "Traditional Dialog Design Applied to Modern User Interfaces" *Communications of the ACM*, October 1990.

[13]Potosnak, K. "Do Icons Make User Interfaces Easier to Use?" I*EEE Software*, May 1989.

[14]Potosnak, K. "What's Wrong with Standard User Interfaces?" *IEEE Software*, September 1988.

[15]Prager, J., Lamberti, D., and Gardner, D.L. "REASON: An Intelligent User Assistant for Interactive Environments" *IBM Systems Journal*, 29:1, 1990.

[16]Rosson, M., Maass, S., and Kellogg, W. A. "The Designer as User: Building Requirements for Design Tools From Design Practice" *Communications of the ACM*, November 1988.

[17]Sutcliffe, A. G. and Wang, I. "Integrating Human-Computer Interaction with Jackson System Development" *The Computer Journal*, April 1991.