DESIGN OF USER FRIENDLY INTERACTIVE INTERFACES

by

JOHN FRANK YORK

B. S., ILLINOIS WESLEYAN UNIVERSITY, 1966

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1982

Approved by:

Major Professor

TABLE OF CONTENTS

LIST OF FIGURES

1.0 INTRODUCTION

The purpose of this report is to examine some of the principles and issues in designing a user-friendly interface for an interactive system. With the expanding use of computers, especially individually owned or used mirco-computers, more and more users are no longer highly trained computer personnel, but rather a cross-section of novice and experienced users. The user-friendly interface is that part of an interface that makes life easier for the user while trying to solve a problem. The design of a friendly interface allows both groups to use an interface, the first learns while using, the second performs the job efficiently.

The first part of this report will discuss some of the principles and issues that are important in the design of user-friendly interfaces. Although these interfaces include machine design of the user hardware, this report will be limited to the interface created by a portable applications program used on some existing hardware. The main design principle is that an interface allows anyone to use it regardless of their level of experience on either computers or the interface itself. Friendly interfaces should be designed or implemented so the novice can enter the interface and use it. At the same time, the experienced user should not be hindered by the aids needed by the novice.

The second part of this report will analyze an existing user interface for its degree of friendliness and offer

suggestions for improving its friendliness. The analysis gives examples on improving friendliness so that all levels of users can interact with the system in an efficient manner. The analysis will apply the principles and issues discussed in the first section to improve the friendliness of this system.

2.0 BACKGROUND

The user interface is that place where human and machine meet. The driver of a car interfaces with his car by using the steering wheel and various pedals. In the domain of computers, the interface for an interactive system consists of a key board, cathode ray tube (CRT), fingers, and eyes. The human gives the computer commands or instructions by typing them in and the machine returns the messages on the CRT. Just like he speedometer of the car tells the speed of the car, the terminal messages tell the user the exact state of the computer. As Schofield [33] stated the user interface consists of "all messages that can pass between the user and the machine and the conditions under which they can occur." Another definition of an interface is one provided by Moran [20], which is that the user interface consists of "those aspects of the system that the user comes in contact with physically, perceptually, or conceptually."

The attributes of user friendliness can not be defined as accurately as the interface. However, the opposite of friendliness, unfriendliness, of an interface can be described much better. As indicated by Gruenberger [9], "user-hostile" can be defined accurately and with a great depth of examples after six months experience in the computer field. Everyone can come up with examples of a user-hostile interface. How does the message "ERROR IN 24567" help a novice user? Even worse, how about just "ERROR"? For many, using a computer is difficult because of the lack of information on which to act. Typically, the

interface is designed for the experienced user and not the novice who is usually forced to fumble through the system using a weighty users' manual and to ask questions from whomever is around.

In summary, by having the machine supply all the necessary information for moving from one command to the next, the user can concentrate all energies toward problem solution. The friendliness of an interface cannot have an absolute value, since the interface is judged for friendliness by humans each having a different opinion of what exactly makes an interface friendly. These opinions are based upon a multitude of factors, all of which cannot be incorporated into the design of an interface. For this report, the friendliness of an interface will be the relative ease by which any person can use an interface without refering to a users' manual or some other non-machine aid. The interface should be fully capable of taking a user entirely through the system on its own without hindering the professional.

3.0 HARDWARE/ENVIRONMENT ISSUES

Hardware includes all physical parts of the computer. For an interactive interface this is basially a terminal producing softcopy used to pass messages back and forth between the user and the computer. Many aspects of the physical design of machines need to be studied. One example is the design of the keyboard, which presents many difficult problems. For example, the location of the characters with respect to each other, spacing between keys, glare of the key, method of activating keys (heat sense, physical pressure, light sensitive), shape of key, slope of keyboard, are all elements for consideration. Hardware design is a complex field in its own right and beyond the scope of this report. An even larger area, which encompasses hardware, is the users' working environment. This too is beyond the scope of this report.

Since interfaces are usually software packages used by many people on a wide variety of computer systems, the interface must be able to be implemented on a wide variety of machines. This implies that the software package will not have any control over the types of terminals used and and the environment in which the package is used.

Another hardware issue that has an effect on a user-friendly interface is the response time of the machine. As pointed out by Shniederman [35], response time of the interface has several effects. First, if the users are conditioned to an immediate response and do not get it they become concerned. The interface should provide some

messages if a particular operation is to take a long time. Second, variability in response time generates poorer performance and low user satisfaction. Consistent response times, up to a certain limit, are more important than just fast response times. Fast response times might even degrade user performance because the user may try to match machine speed and fail to check input leading to even more errors and frustration.

4.0 PSYCHOLOGICAL ISSUES

An examination of the psychological issues is necessary in the design of an interface for friendliness, especially if the interface is used by people with various levels of computer experience. Many psychological issues are involved in the interaction of people with a computer. This report will examine those that are important in the design of an interactive interface for use by both novice and experienced users. The user's experience with computers in general and with a specific interface. Just as hardware issues could not be included in this report because they were beyond the control of the software package, there are certain psychological issues that cannot be included in this report. For example the software package creating the interface cannot have any control of the working environment existing in the office where the interface is used. It cannot provide positive or negative reinforcement for the users other than that displayed upon the CRT.

There are six psychological issues that have an impact on the design of an interactive interface. They are memory of users, level of experience, closure, attitude, anxiety, and control. Each of these subjects will be discussed in the ordered listed. Other areas concerning the psychological design of an interface are included under these six.

4.1 MEMORY

The memory capacity of the user has the greatest impact on the design of a friendly system. Since the interface provides all the information about the software package, the only way the user can be given information about the system is on the CRT. The CRT cannot hold the commands or instructions on the screen while the user goes through the system. Only that information which is held in the user's mind and displayed on the CRT will be available for determining the next instruction. Therefore, the interface has to be designed so that it does not require more memory capacity than the user can apply, or it must have a way to page back through the commands like a user's manual.

The first source of information is in sensory information storage. Sensory information storage has a very short span of retention and is constantly being bombarded with input from the six senses. These raw sensations are held only for the time necessary for short-term memory to examine them and either hold or ignore them. Short-term memory processes the information in the sensory information storage. Short-term memory can retain information for a longer period of time but is limited in the number of items that can be retained. An item of information can be retained for a longer period of time by "rehearsing" (Lachman [13]) it. Rehearsing is the process of repeating the piece of information over and over again. An example is repeating someone's name after being introduced to them the first time.

Long-term memory is where the user's permanent information is kept and can be retrieved and used by the short-term memory. The process by which the human mind places information into permanent storage is not well understood. However, it does appear that the placement of information into long-term memory requires time and effort. The actual organization and continuing reorganization of information in long-term memory is even less understood (Sheiderman [35]). The method for moving information into long-term memory appears to be a process aimilar to that of moving information from the sensory information memory to short-term memory - rehearsal or repetition.

The rehearsing or repetition of information for retention is important in designing an interface. George Miller's 1956 paper, "The Magic Number Seven - Plus or Minus Two", indicates that the short-term memory of a person can only handle seven "chunks" or units of information. Each of these chunks can become larger as the person gains more and more experience with the information. Short-term memory can transfer these chunks to long-term memeory and then retrieve them, combine several together, and transfer them back to long-term memory over and over again. Short-term memory, with its capacity to handle only seven chunks or units of information at one time is what Lachman [13] terms the "bottleneck" in the human information processing system. This bottleneck limits learning by the user. Short-term memory cannot be overloaded and this is a serious limitation in designing the user-friendly interface. The user can only handle approximately seven units of information at one time;

but time and repetition can compress several commands into one chunk. Information from both sensory and long-term storage can be retrieved by short-term memory. Short-term memory with its ability to work with information from both of these memories is sometimes referred to as working memory. With the information from these two sources, short-term memory forms a conceptual model of the interface while learning it by creating a single chunk from several other chunks. These compressed chunks represent a conceptual or semantic meaning to the user. It is only through short-term memory that the user learns to use the system. Moran [20] found that a conceptual model was formed by users of his Command Language Grammar (CLG). He also found that the conceptual model formed by the users was not in all cases the model that the designer had in mind. This problem, the user forming a different model than that desired by the designer, was also noted by Gaines [7].

Another limitation stemming from the limited capability of short-term memory is that people tend to get confused using an interface. While studying users of ZOG, a menu driven network selection system, Robertsen [30] found that people failed to read all of the menu when displayed on the CRT because they could not hold all of the information presented on the menu in short-term memory. Since the information for moving through the system was lost, the user soon became lost within the system.

4.2 LEVEL OF EXPERIENCE

The level of experience of the user is based upon the user's prior training or familiarity with computer systems in general, and the specific interface being examined. Undoubtly the novice needs to have available more detailed instructions than the experienced user. However, even the experienced user, in some respects, acts just like an inexperienced user when exposed to a new interface. Ledgard [16] found support for this issue when he was looking at natural languages for interfaces. This happens because users move back and forth between the novice and experienced levels as they learn the old system and start on another. For this reason users experienced with computers will not be disturbed by learning on an interface designed for the novice. The experienced user will, however, get frustrated quickly if it is necessary to follow the novice's repetitious dialogue, especially at low terminal speed. Tagg [39] pointed this out in his work. Conversely the novice cannot be expected to learn efficiently on an experienced user's interface. The novice must be given the opportunity start at a simpler level.

Anything that can be done to reduce the amount of new information required to operate the system will aid the user. Using a natural language in the command structure for the interface automatically gives the novice an advantage for there is no need to learn new words or symbols. However, long commands in a natural language slows down the experienced user unless there is a way to shorten the

commands such as only using the first letter of a command. In planning any design, the needs of both the inexperienced and experienced user must be taken into account or the interface may not serve its function.

4.3 CLOSURE

Closure is the relief a person feels when information in short-term memory no longer needs to be retained. Closure creates a strong drive to complete a task, free short-term memory, and gain relief. Every time you sign off from a computer or finish typing in a command, closure is experienced. Novice computer users will experience a greater feeling of relief than an experienced user in entering the same command. The experienced user will be able to use larger and longer commands because as experience is gained, more powerful commands are needed to produce the same amount of closure.

The pressure of closure means that users, especially the novice, may prefer several small commands rather than one large one (Shneiderman [35]). The size of the command or entry will, of course, vary with the feelings or preceptions of the user. The desire of the novice to use several small commands is twofold. First it allows frequent checks at each step of the operation to gain assurance that all is going well. Second, it permits the user to forget about the earlier portions of the operation, thus alleviating short-term memory requirements.

4.4 ATTITUDE AND ANXIETY

Attitude and anxiety are closely related and affect each other. Attitude is the state of a person's willingness to learn. Anxiety is a feeling people have when placed under stress. As outlined by Lindsay [15] stress has three causes:

a. Internal models are inadequate to explain the present situation.

b. Internal models lead to an undesirable result that a person feels powerless to prevent.

c. Stress itself.

People's attitude toward a computer is partly explained by the fact that they want to avoid situations that might produce anxiety or undesirable outcomes. If the user has a pleasant experience with a computer, then there will be a favorable attitude toward the computer. If there is an unpleasant experience, a negative attitude will result. Reducing the users' anxieties will improve their ability to use the computer. Smith [36] stated that one reason the computer created anxiety was that its workings were invisible. He further stated that a person just introduced to a computer exhibits emotions based on fear, awe, and general uncertainty.

Users with negative attitudes toward computers make more errors and learn more slowly than those with positive or neutral attitudes. Lucas [19] found that the use of his interactive information storage and retrieval system for medical research was significantly associated with favorable

"user attitudes". Individuals who rated the system highly tended to use it most frequently. Overcoming a negative attitude is partly the job of a user-friendly interface. The design of the responses can improve the attitude of the user toward the computer. A negative attitude displayed by the user is based upon the anxiety generated from working with the computer. To overcome anxiety and any attitude problems, every effort must be made to make the user feel at ease when confronting the terminal and using the interface. Smith [36] stated that communicating with a computer is typically a very "unsatisfactory experience...with rather clumsy devices". Communicating with a computer is, according to Hayes [10] a "time consuming and frustrating experience". On the other hand, the user will not be appreciative if the interface is patronizing.

The user will feel more comfortable with the interface if the instructions and messages to and from the machine are written in clear, concise English and are easy to understand. Simple tasks should be the norm until the user gains confidence from their successful application. Diagnostic messages should be understandable, non-threatening, and low-key. Constructive messages and positive reinforcement produce faster learning and increase user acceptance more than short, terse, cryptic messages that force the user to go to a thick manual for meaning. Robertsen [30] found that by allowing the user to remain at the terminal was good in that it eliminated the jump from barely acquired knowledge to actual use and any search for relevant information by having it available on the

terminal.

4.5 CONTROL

Control is a psychological issue in the design of a user-friendly interface because too often designers have attempted to design the interface so that the computer is an actual entity or intelligent being. Control is the sense of being in charge or command of an object such as a tool. This tool responses to the wishes of the user and not the other way around. Control of the machine cannot be taken from the person using the interface. The interface demanding answers or commands from the user may satisfy the novice user. Once the user becomes familiar with the interface, the user will want to control it and not vice versa. In Dzida's [6] survey of user perceived qualities of a friendly interface, control was one of the factors repeatedly mentioned.

As users gain knowledge and maturity, they begin to resent any attempt on the part of the computer to demand commands from them. Experienced users regard the computer as a tool. They resent messages that even suggest the idea that the computer is demanding responses from them. Control also means that there must be no doubt in the user's mind as to who made the error. The computer is just a tool which reacts to the user's commands. Both of these points were brought up by Gaines [7] in his discussion of his interactive dialogue programming rules. He also stresses the point that users want to dominate the computer. Every

activity of the system must be a clear consequence of the user's actions. Popularity of micro computers is partly due to the fact that the user/owner feels in control. He can see, feel, and handle the file kept on a floppy disk totally unlike a file on a mainframe.

4.6 ERROR HANDLING

Creating the most concern with the users, especially the novice, is error handling. Error messages in computer systems have always consisted of short cryptic messages that can be totally confusing. Ling [18] suggests that messages have little value in an interactive system if the diagnostic message contains little chance for easy recovery or change of tasks. The interface must be tolerant of the user no matter what level of experience. One example of this is the novice who enters the system with great reluctance, and as the next command is being entered the system signs off with a terse, abrupt message stating that it had waited long enough for the command. It is doubtful that this person will ever attempt to use the computer again.

Error handling can also bring about negative reinforcement. Sounds or noises associated with a error message have always proved to be ineffective because the sound draws the attention of others to the user's error. No one likes to have attention drawn to their errors. Any error message that forces the user to leave the system to find the meaning of an error message will decrease its effectiveness. Doherty [4] states that the user's time is

valuable and the computer should be managed so that the user's ability to work is enhanced with the "least amount of inconvenience."

Error handling impacts on all the other psychological issues discussed earlier. The anxiety and attitude of the user is greatly affected by error handling routines. The naturalness of the error message impacts on the ease with which the message is understood. In describing SITAR, which is an interactive text processing system for small computers, Schneider [32] stated that it tried to prompt users and lead them to successful completion of their session time rather than just tell them that an error had occurred.

5.0 DESIGN ISSUES

Design of an interactive interfaces will, in the future, incorporate all aspects of computer programming, hardware capabilities, access control, data base design, and user capabilities. Rayner [25] states that data bases of the future will have to consider open systems with a wide variety of users. The design issues covered in this section can be used by a software package without considering any hardware restrictions or capabilities.

The most important thing in designing any user-friendly interface is that there be a deep-rooted concern by the designer for the convenience of the user. The degree of friendliness of the interface must be defined early in the design stage. A philosophy for designing business programming languages as given by Zloof [41] is that the user should be required to know very little about the system to get started. Lucas [19] supported this philosophy with the comment that "too often technical issues become control focus" while the reactions of the users are ignored.

Compounding the problem of designing what the system should give the user in the way of friendliness is the differences in levels of experience of the users. In other words, the designer must either know the level of experience of each group of users and build specific interfaces for each, or design an interface that will meet the needs of many users. Unfortunately, in the past, the design of the interface served only one level of users mainly because of time and money constraints. One other reason for the poor

design of user-friendly interfaces is due to the fact that the computer-man dialogue is poorly understood; as pointed out by Bernard [1].

A well designed interactive interface can relieve the user of the administrative details of learning the interface. Some of the administrative work could even be incorporated in a programming language as proposed by Negus [21] in his DIALOG language. The user interface that can save a person time will be extremely valuable in the future and will expand the use of computers.

One of the main problems in designing a user-friendly interface is that there are few hard and fast principles. Everyone has a specific example of what the interface should not do, but very few have examples of what it should do. What is friendly for one person might be extremely hostile for the next. Unfortunately data for the design of a user-friendly interface requires observations of people which are not easily quantifiable. The design of a user-friendly interface is still very much an art. It will take design and testing to find out the "best design". In several articles the authors (Moran [20] and Barnard [1]) have stated that designing a user interface is still a "purely intuitive endeavor" and not something that can be put into laws. In addition sometimes the needs of the users change through the design. Several studies have been done by Reisner [29], a psychologist, which explain how several psychological principles can be applied when designing a friendly-user interface.

Design issues are broken down into three areas: User

Issues, Machine Communication, and Error Handling. This breakdown shows the functional areas of the interface exchange with the exception of error handling which is a special area of machine communication which needs a section alone to fully explain the issues involved. Machine communication includes the format for messages between the user and the computer.

5.1 USER ISSUES

The first issue is the problem of determining the level of knowledge or experience of the people using the interface. Some effort must be made in collecting information about the range of experience of the users. The more the experience, the fewer the basic instructions. The designer will have to obtain background information on the office or company that is going to use the system. This information can be used later by the designer to insure that the model of the system which is formed from the interface is understood by the intended users. There is always a tendency on the part of the designers, because of their familiarity with the interface, to not make it simple enough. This was discussed by Sneeringer [37] in his case study of designing a text editor. What the designer thinks is simple may be totally confusing to the user.

Determining the degree of knowledge of the user might be as simple as insuring that all the keys on a computer key board are explained if they are not usually found on a standard typewriter. In his article, Palme [24] states that

with the touch method of typing, letters are easier to hit than numbers and so the way to select commands off a menu is by typing in letters and not numbers which are higher up on the key board. Simple things like this can be overlooked by the designer unless there is a full appreciation of the level of knowledge of the users.

Another area of interest in the design of an interface is the depth of detail required for commands and messages. This affects three areas: the type of command format for the system, error messages, and the prompts or return messages from the system. The commands the users gives form the user's perception of the system. If the experienced user are forced to utilize a very detailed and time consuming menu selection format to give commands, their perception is that the interface is too slow and clumsy for efficient work. In designing the interactive interface for the Statistical Package for Social Sciences (SPSS), Tagg [39] found that the sophisticated user gets upset if commands were picked from a verbose menu. The ability of the user to choose the level or type of command formats permits the user to precede at a self-determined pace. This will allow the experienced user to give more powerful and briefer commands than the novice. Instead of paging through several layers of menus to finally complete an operation, the experienced user can enter the whole command on one line using abbreviations and contractions. Adaption of this capability to contract or expand the number of entries to perform an operation lets users seek their own level.

Messages have an affect on the attitude of the user. If

messages cannot be understood, there is less chance that anyone will attempt to enter the interface. If it is too difficult to use, it will not be used. As in the development of CLG, Moran [20] noted that it is not enough to provide a powerful tool if the user cannot use it. The designer must realize that the users are very aware of their time. They might not appreciate the elegance of the design when they are forced to spend several hours of their time in an unproductive effort, in their minds, learning how to use the interface.

When designing the interface the memory capacity of the users is of prime importance. Anything that can be done to reduce the burden of the short-term memory aids the user in working with the system. Increasing the similarity of the commands to a natural language so that the user does not have to learn a new language accomplishes this. Ledgard [17] found that the people using a text editor with English commands did better than those using a regular editor. The design that allows the user to apply that which he already knows to the interface will also increase the rate of learning relative to the interface. The novice may not be able to apply very much, but the experienced user can apply general computer knowledge to understand the system. As pointed out by Reisner [27] the experienced user may attempt to apply rules from another interface on a new interface and become just as confused as the novice unless the interface allows for a graceful exit or completion of the task.

Allowing the user to determine the type and level of messages that are used in the interface can reduce

frustration and anxiety. As indicated by Smith [36] communication with a computer is at best a very unsatisfactory experience which depends on very clumsy devices with the onus for error checking on the user. It does not make sense for the novice user to try to use a macro command about which there is no understanding or comprehension. Adjusting the command format to the degree of the experience of the user extends to messages from the machine.

One way to choose the type of command format is to ask the user at the beginning of the session to pick what type of format is desired. Of course, the user should have the option of changing the orginal selection as experience with the system is gained. The command to receive aid can be implemented by having the user type in the command followed by one or more question marks which determines the amount of detail concerning the command. This way the novice can generate very detailed descriptions of the commands, while the experienced user can get very short messages but still retain the capability to get more if needed. This would be very helpful to the casual user.

Adjusting the interface to account for the user's short-term memory limitations is a very real problem. As discussed earlier a bottleneck in the machine-human interface is the short-term or working memory. Since short-term memory can only hold seven chunks, anything with more than that will cause confusion or a slow down of the learning process. As discussed earlier, these seven chunks are not the same for every user. Experienced user's chunks

contain more information than the novice's. The key point for designing an interface for a variety of users is the ability of the system to accomadate several different levels of users. Persons with a knowledge of the command structure can easily condense a sequence of commands into one semantic chunk, but the novice must take each command one at a time. Therefore, if the experienced user is to become efficient by using all of short term memory, the interface must permit more than one command to be entered at a time.

Every effort in designing an interface should be directed to the end of letting the user enter commands or instructions in a small enough portion that can be retained in short-term memory. This means in practice that the method of providing prompts should be such that the user is not required to retain too much information in short-term memory. For example, it is better to have three separate menus displayed on the CRT, one at a time, rather than having one menu containing all of the input required. First, this limits the amount of information that must be retained and relieves the user of any anxiety caused by entering a long command. Second, the user is prevented from making any great mistake from which recovery is impossible. The user will also feel more comfortable using smaller steps which result in a greater chance of job completion and not an error message.

The use of a natural language in the design has been discussed before. Natural language helps the novice the most. Since the computer is being used more frequently by the untrained user, the choice of a natural language for the

messages that pass back and forth between user and machine becomes more appealing. The novice is not the only one helped by a natural language because it also supports the experienced user. It seems that in the use of a text editor, the group using the version written in natural English did better for all levels of experience than the group using cryptic messages.

Use of a natural language lets the user rely on an already acquired knowledge in communicating with the interface. A natural language frees the user from learning the terse, symbolic representations so common in computer systems. Of course, the problem with any natural language is that it is very imprecise and difficult to translate into correct system commands. A natural language also contains ambiguous statements and commands. Hayes [10] notes that the problem in communicating with a machine is that people are too used to communicating with another person in a verbal manner. Some of their errors are automatically corrected by the listener during the course of a normal conversation. In addition the listener eliminates those parts of the communication that have no meaning or are not needed to convey the semantic value of the conversation. The interface on the other hand must recognize every symbol or lack of symbols and attempt to translate them into a possible change of state.

Consistency and uniformity make it easy for the user by reducing the short-term memory requirements. Sabine [31] points out that if the commands follow a natural English syntax, they are easier to understand. Consistency is also

achieved by keeping the same relationship or format for every command. The command syntax can always have the parameters follow the command, like English has the object follow the verb. As described in Moran's article [20] consistency is fully designed into CLG which was developed by his group and described as levels of understanding between the machine and the person. Each level produces another level of meaning to the user and depends upon the language for understanding.

Consistency of commands can be implemented by positional formatting as used by Bernard [1]. Positional formatting has the commands and their parameters follow the same sequence throughout the interface. For example, the first word is always the command itself followed by the options available. In the case of an editor, the command for changing the line of a text would always be followed by the line number and then the change, correction, or addition. The command and its parameter cannot be entered in any other way.

Uniformity is achieved by insuring that the interface always has the same way of forming contractions for commands and for progressing from one command to the next. This does not say that the user must input the same set of commands each time he wants to go from one state to another. The system only allows for one sequence of commands to get from one state to another. An example of this is the technique of parallel versus sequential execution of commands. With sequential commands the user must enter each command or parameter on a separate line. Sequential commands force on

the experienced user the long boring task of going through several layers of commands. Whereas if the system accepted a series of commands on the same line, the interface could change states by parallel execution.

Another aspect of reducing the memory load of the user is having all the information that the user needs to operate the system available at the terminal. Conceptually, this is more than just allowing the user to use help commands. It means that the user should be able to stop giving commands or pause in his task and call up a definition of a commmand or a detailed description of the next command without cancelling the work done so far. The effect is to reduce the mental effort of the user in trying to think up the next command and to save time by not having to leave the terminal or exit from the interface to look up the answer in a user's manual at some other location. Looking up the correction immediately at the terminal permits the user to obtain the necessary information without breaking up a train of thought. It also has the user learn without an instructor.

One final aspect to consider is the effect of a small change in the users model on the interface. Users form a conceptual model of how the interface works in their minds. As stated earlier, this model may or may not represent the model the designer had in mind. In any event, this model is very important because it is what the user solves problem with. When the user makes a small change in the conceptual model by changing a command, there should be a correspondingly small change in the operation of the system.

Macro commands developed by the user are based on the user's conceptual model and expands the problem solving capability and facilitates the thought process of the user.

5.2 MACHINE COMMUNICATION

Machine communication is concerned with the replies that the machine displays in response to commands, and the information contained in these displays. Natural language plays an important part in machine replies as it did in designing the commands with one important difference. The machine does not have any limits as to the richness of the vocabulary and syntax because it does not have to worry about having to translate every symbol into action as commands are. There is no reason why the messages from the machine cannot be as detailed as the user wants or needs in order to accomplish the job or to learn the system. An area of the machine responses include error messages which are very complex and are discussed separately. Machine communication will cover prompts of the interface for the next command, methods of dealing with command selection, help facilities, and depth of help facilities, and designing so that the user feels in control of the computer.

The first issue in machine replies is that the interface should guide the user to successful completion of the task. To accomplish this many help messages and detailed descriptions of the interface commands are needed. As pointed out by Doherty [4] the terminal is so much more than a typewriter. It should be able to provide the user a vast

amount of information about the system so that the user can traverse through the system with a minimum of effort.

The need to lead the user through the system must be tempered by making the user feel in control and insuring that the user knows this. As discussed before, the novice may feel satisfied with being lead through the system command by command, filling in the blanks, and answering the machine's prompts. The experienced user becomes very upset if forced to do this. In short, the interface should only provide the prompts that are absolutely necessary and not any more. One way of doing this is to have the user select the level of prompts desired. Now the experienced user gets only detailed prompts when working with a part of the interface that is unfamiliar or has not been used in a long time. The novice can start with detailed prompts and use them until experience and satisfaction with the commands is gained to transition to shorter prompts. User control is extended by allowing for interruptions in the current task to start or resume another. With the interruption the user can stop the present task, go to another task or obtain information about the next command, and then go to the same spot in the orginal task without having to re-enter the interface. Right along with this is the capability to corrects commands whenever necessary. In addition, the user should be able to develop macro commands whenever desired. Doing this will allow the user to fully develop a model that will match the user's thought process and increase the user's problem solving ability.

The first effect that the machine replies will have on

the users as they use the system is on their attitudes. If the replies are incomprehensible and hard to understand, the system will not be used. If not used, the system is useless no matter what its applicability. The interface should as Moran [20] stated help the user without getting in the way. The interface must be efficient, easy to use, and easy to learn. The messages which the machine provides are the only way the user knows what it is doing. In many ways the design of the text of the messages is a purely intuitive endeavor.

Based upon the messages from the machine, the user will build a model of how the commands are accomplished. The interface's commands allow the user to assimilate the commands to form this model by inducing its behavior. This model will determine how the user will apply the system and whether or not the user will be able to include all system features. One of the points brought up by Doherty [4] is that the interface should work with the least amount of inconvience to the user. Complexity is wasteful of the user's time and can be reduced by having the machine do most of the work.

Next is the amount of detail in each message and who should receive which message. The main task here is to take the user through the system without hindrance. The level of detail in the message will be based upon the corresponding level of experience. This has been discussed several times before. The point is that both the novice and the experienced user will need detailed messages at certain times. The most detailed prompt is the menu and is the type

of message that the novice will be working with. The experienced user will be slowed done if forced to use menus.

The next level of prompts is the single line or fill-in-the-blank prompt. The single line prompt is a single line containing the command and a list of its parameters. The user simply picks the command desired and types the necessary parameters behind it. No explanation of the command is given. This type of prompt can be done two ways: parallel or sequential. In sequential prompting each command and each parameter is listed on a separate line. This the slower of the two but does allow more information to be placed on the screen. With parallel prompting all the commands and their parameters are entered on one line. This is faster for the experienced user but is very confusing if the user is not totally familiar with the interface. The most experienced user may not even need prompts of any sort.

The point where the user picks or is given the level of prompts should be early in the program so that time is not wasted. On the other hand if the user chooses too high a level of prompts, the interface should lead the user back to the simpler more detailed prompts without damage to the work already accomplished. In any event, the simpler the command structure or selection method the more useful the interface will be. Positive reinforcement should be given to the user whenever possible. After every command the user should receive some sort of acknowledgement that the command has been received and the computer is processing it or ready for

the next one. The display of the next prompt shows the user that the system is ready for the next command. The prompt of "very good that is a correct command" will quickly create acrimony toward the interface. A more positive response would be "++" for a correct command that the system must take time to work on before the entry of the next command. This response shows that the command is correct and that there is not going to an error message. As discussed by Shneiderman [35] the user will "resent any message that suggests to him that the computer is in charge...". Additionally the interface should not display many menus or prompts asking the user to verify a command or else the user will stop reading them and automatically give a positive response without checking the command. This was noted by Sneeringer [37] in his study of the text editor, OCCAM. If conformation is necessary to prevent unrecoverable, destructive changes, the interface should check the whole command before asking for confirmation; otherwise, the user will become upset about making several entries to get one correct command.

5.3 ERROR HANDLING

Error handling is a special area of machine messages that must be considered separately because of its impact on the interface. A good design including helpful prompts and extensive learning-while-using messages will have need for few error messages. Error messages affect the user the same as commands and prompts. There are two aspects of error

handling for which the interface must plan. First, the error procedures must protect the user from making serious mistakes that cause the loss of large amounts of work. Second, error messages must tell the user exactly what the error is and how to continue on.

Error messages have an even greater need to be understandable. They must be written so that they do not create any ill feelings. The error messages need to be objective as possible and constructive in that they teach the user something about the system. Error messages must also be concise yet understandable as was previously stated. Natural English should be used at all times except for the most experienced. The level of experience will have an effect on the content of the message and is discussed later.

As outlined by Ling [18] in his article, the interface must assume that the user will make errors all along the sequence of commands and check each place for the error and be able to produce a meaningful error message at that point for the user to correct the command. An approach taken by Wilcox [40] is that the computer should not attempt to "recover" from errors but rather the interface should inform the user of the error, and attempt to give prompts that take the user away from it. This might be a matter of displaying a menu describing possible ways to proceed. Dzida [6] found that fault tolerance was a very important factor in measuring the friendliness of an interface but as experience increased the importance of fault tolerance decreased.

The interface should allow the user to select the amount

of detail in error messages just as in commands and other messages. The more experienced, the shorter the messages. The interface can examine the type of commands being entered and give corresponding error messages. The longer the command, the more verbose the error message. If the direction that the user wants to go cannot be determined, the interface can always reply with the most detailed message.

Another feature of error handling is to allow the user, in the event of an error that causes the machine to fail, to return to some known starting point so that the whole task is not lost. This can also be very important in case the computer system goes down. The interface can also check for spelling errors and try to correct them or ask for confirmation if the intent of the user is not clear based upon the previous command. Along with this is the capability of the machine to require only partial retyping of errors in a command. This was suggested by both Dzida [6] and Ling [18]. If this is not possible some indication of the exact position of the error should be given. Shneiderman [35] brought up the point that human performance improves if the error messages are issued immediately after they are found. The advantage of having the interface check each character as entered is that the correction only requires retyping of that one character and the not the whole line.

Error handling is also affected by the user's desire to control the computer. In this respect, it is very important that the error messages convey the fact that the error was

a direct result of the commands given by the user and not the fault of the computer. Avoiding blame as described by Gaines [7] is making the activity of the system a clear, consequence of the user's actions. The user cannot blame the computer of being wrong. This is part of giving the user control of the interface. The error messages should convey to the users in simple, clear words that the error was a consequence of the their actions. The message should do this in a polite and objective manner and can be a positive learning technique if phrased in the right manner.

The interface must also layer the levels of help messages. The simplest and shortest description of a command should be the first available for requests for help. When the error is detected the user looks at the error message and then decides what level of help message is desired. The method used by Scholfield [33] for MM/1 was a series of question marks after the command for more information. This gives the novice and the experienced users flexibility in working with the interface.

The interface should also protect the user from costly mistakes or errors such as in a time-sharing system where files cannot be recovered. The interface can require the user to think again about any command where this may occur. As discussed earlier, confirmation cannot be asked too often or the user will automatically type it in.

Lastly the error messages must provide a positive environment for the user. An example was given of the person who was thrown off the system the first time on which created a very negative attitude toward computers. If the

error messages are helpful to the user in a positive way and encourage the user to correct mistakes, a favorable environment is created and the friendliness of the system is increased. The user should never be left hanging somewhere without knowing where to go next or what can be done to get out of the present command. Schofield [33], talking about MM/1, stated that an "escape" from an error should be a standard feature on all commands.

5.4 SUMMARY

The design of the user friendly interface requires consideration of many factors, all of which are not under the control of the designer. The design of any interface is still an art and not an exact science. With a dedication to creating a truly user-friendly interface, great improvements in user-friendly interfaces are possible. If the designer is aware of even a few simple psychological issues of human behavior, the interface will be better and more useful to the user. User-friendly interfaces will become more important as computers are used by more and more non-computer people.

6.0 ANALYSIS OF USER INTERFACE FOR FRIENDLINESS

6.1 DESCRIPTION OF INTERFACE

The following section will analyze an existing full screen editor called SEDIT for user-friendliness. SEDIT provides a visual machine interactive means to move, correct, change, and manipulate displayed text files. Friendliness as defined earilier is the ability for any user with any level of experience to immediately start using SEDIT without referring to anything but the CRT. To be this friendly, SEDIT must be able to take any user through the entire system without any reference to outside aids and yet not hinder any experienced person with unnecessary prompts. Since SEDIT is intended to be a software package that can be purchased by any company or organization with several different levels of users, it is very important that it be as friendly as possible. A wide range of user experience means that SEDIT has to quickly teach the novice how to become productive. While learning the syntax and commands, a conceptual model of SEDIT is formed. With the model established, abbreviations and short-cuts to gain speed are used. There will be many casual users who have already formed their conceptual model but who will need help messages or aids throughout SEDIT to refresh their memory on unfamiliar commands. These people can receive the same messages as the novice but only when needed.

With a software package being sold for implementation on a wide variety of machines, there are several friendly

features that cannot be implemented by SEDIT. The first is the capability to check each symbol as it is entered so that the user is not forced to retype a command because of one mistake. Another is control of the response time of the machine. As suggested, a constant response time is better than a variable response time within certain time spans; however, this can be fixed only after extensive testing and evaluation. Of course, all aspects of the user environment at the work station is beyond the control of SEDIT. This includes such things as the physical terminal, its screen, keyboard, and work area.

In summary the only effect that SEDIT has is on the messages that pass between the interface. This analysis of SEDIT is limited to the design of these messages and the conditions under which they occur. SEDIT is a relatively simple interface in that there are few commands available. However, SEDIT still requires all the design principles discussed earlier.

6.2 OVERALL DESIGN

The overall design of SEDIT is directed toward one level of user, the intermediate. SEDIT has only one level of messages for both commands to and messages from the system. Most of the commands produce a prompt listing the parameters or commands for the next state without any way to type in multiple commands. These are aspects of SEDIT indicate that it is an interface directed toward a person with some experience. Without any help messages or error recovery

messages, SEDIT can not be used by the novice. Therefore, SEDIT is designed for a level of user between the two.

Several friendly features have been left out. First if the user becomes lost or confused about the next command or wants to leave to perform some other task, no way exists to do so. For example, if the user wishes to end the session but cannot remember the command, SEDIT does not have any way to exit without aborting the present session and starting over again. A way to handle this is to have a HELP procedure which displays all the commands followed by a short definition. Second, any person attempting to use SEDIT without a manual will get lost almost immediately. SEDIT cannot lead users from one command to the next and teach the commands while doing so.

The following sections will discuss specific design considerations of SEDIT for improvement of its friendliness. Starting with entry into the system, the sections will then go through the commands and finish with the error and help messages.

6.3 ENTRY INTO SEDIT

Entry to SEDIT can serve several purposes. First it can determine the level of experience of the user. A relatively straightforward procedure has a menu ask the user for the level of commands and prompts wanted. With this knowledge, SEDIT simply accepts only those commands for that level until told to do otherwise by either the user or the system. A more subtle way requests the first command while

including the command to continue. If the first command is an abbreviated or terse command then SEDIT accepts any level of command. On the other hand, if the command is the one displayed on the menu, only complete commands are accepted. Requiring the novice to use the whole command builds up the conceptual model. Figure 1 shows how the type of command given would produce different menus for three levels of commands.

The first prompt or menu shown should be brief so as not to slow down those with experience and yet long enough to provide information to the novice. Figure 2 performs the function of determining the level of experience without slowing down or agitating the experienced user with a long menu that will be seen hundreds of times.

	COMPLETE	VERBOSE
	COMMAND	MENUS
ENTRY	ABBREVIATED	SINGLE
MENU	COMMAND	PROMPT
	DIFFERENT	TERSE
	COMMAND	PROMPT

Figure 1 Determination Of User Experience

SEDIT

TYPE IN "SEDIT" OR FIRST COMMAND AND "RETURN".

TO LEAVE SEDIT TYPE IN "QUIT" AND "RETURN" AT ANY POINT.

Figure 2 Entry Prompt for SEDIT

Presently SEDIT allows the experienced user to quickly access the last file updated or worked on. An addition of a menu as shown in figure 2 does not slow down the processing because the same command gets to the same state as before only now the novice is forced into giving long commands. Figure 3 shows the three possible types of commands SEDIT will accept from the entry menu.

EXPERIENCE LEVEL	COMMAND
NOVICE	CONTINUE
INTERMEDIATE	C
EXPERIENCED	FILENAME.EXT

Figure 3 Commands Accepted From Entry

Second, the entry point selects the level of prompts displayed. Failure to enter an abbreviated or some other command produces detailed menus which lead the user through the system. These will slow down the user but force learning the system by repetition. These menus show commands that are available and explain their function. Care must be taken so not too much is shown or the user does not read the whole menu. An example of the first menu after entry for the novice is at figure 4. Note that the explanation for the HELP command is given at the end. This information should be given at the end of several menus throughout the system. The menus for the intermediate and experienced users are at figures 5 and 6, respectively.

SEDIT CAN EDIT AN OLD FILE OR CREATE A NEW ONE. ENTER ONE OF THE FOLLOWING COMMANDS:

"CREATE" WILL MAKE A NEW FILE.

"TYPE" BRINGS A CREATED FILE TO YOU
SO CHANGES CAN BE MADE ON IT.

Figure 4 Novice Instruction Menu

ENTER FOLLOWING:

"C" TO CREATE

"T" TO TYPE

Figure 5 Intermediate User Prompt

"C" OR "T"

Figure 6 Experienced User Prompt

The third function of the entry menu is to start an error counter for the number of errors during a session. If the number of errors passes a certain number, the interface automatically starts giving the novice prompts. This will have two effects. First it will speed the learning process by stopping the user from just typing in commands in an attempt to find a way through the system. Second it rewards good performance by not displaying the novice menus. Once forced back to novice menus as a result of too many mistakes, greater attention will be given to the commands. Help calls are encouraged in that if the correct command is not typed in novice menus will be displayed. As discussed later, HELP commands are more than just a list of one or two commands. HELP menus give definitions, describe functions or operations, and more. Displaying several SEDIT commands on a single menu over and over again increases the speed of learning them. A prompt indicating that novice menus will be displayed is at figure 7. Actually the interface can provide two levels of defaults by having SEDIT drop from one level to the next. Short one line prompts are very effective for the intermediate or casual user.

A final note is the need for the command menus to explain how to gracefully exit the system which for SEDIT is QUIT. The HELP command needs to be included in several of the command menus for the novice. Definations of both should also be on the menus displayed at the end of the command. Both is shown at figure 8 which is an example of the CREATE menu for the novice.

TOO MANY ERRORS IN COMMANDS HAVE BEEN MADE.
MORE DETAILED MENUS WILL BE DISPLAY UNTIL
RE-ENTRY INTO SEDIT.

Figure 7 Menu When Too Many Errors Made On Machine

ENTER THE NAME OF THE FILE YOU ARE GOING TO CREATE. AN EXTENSION USED TO FURTHER IDENTIFY YOUR FILE IS REQUIRED. THIS EXTENSION CAN BE EITHER .PAS OR .TXT AND IS PLACED DIRECTLY AFTER THE NEW FILE NAME. THE GENERAL FORM FOR IS

<NEWFILENAME.EXTENSION>

AN EXAMPLE FOR THE NEW FILE LISTINGS IS SHOWN BELOW.

LISTINGS.TXT

AN ENTRY OF "QUIT" WILL ALLOW EXIT FROM THE INTERFACE.
AN ENTRY OF "HELP" WILL SHOW COMMANDS AVAILABLE NOW.

Figure 8 Novice CREATE Menu

6.4 SEDIT COMMANDS

Two prompts in SEDIT show the different commands allowed. A prompt affecting the movement of the cursor is not present but is purposed at figure 9. The first prompt displayed once a file has been located lists several commands and a question mark which is typed in to display the second menu. The second menu lists more commands. Both do not provide any information concerning the function of the commands. An experienced person could possibly guess at some of the functions while the novice could not. The insertion of several menus before these two could provide more information to the novice while not hindering others.

COMMANDS TO MOVE CURSOR TO DESIRED POINT IN TEXT.

COMMAND	EFFECT
<CTRL>H	MOVES CURSOR LEFT
<CTRL>L	MOVES CURSOR RIGHT
<CTRL>K	MOVES CURSOR UP
<CTRL>J	MOVES CURSOR DOWN
"<"	CHANGES DIRECTION
">"	CHANGES DIRECTION
<SPACE>	MOVES DIRECTION
<BACK-SPACE>	MOVES LEFT
<RETURN>	BEGINNING OF NEXT LINE

Figure 9 Menu For Movement Commands

For another layer of prompts for the novice, a functional grouping of commands is needed. Several of the commands, SET, ADJUST, MARGIN perform administrative functions such as setting the margins of the output. INSERT, DELETE, REPLACE, COPY, and EXCHANGE change or replace the text of the files. Several other commands have miscellaneous functions such as examining spelling, verifying or displaying the text on the screen, and showing the hexidecimal location. Adding one menu which groups the commands under a functional heading and asks which group is required reduces the number of commands displayed on the screen. For those already knowing the next command, the command is typed in and the function performed and the menu skipped. An example of this sequence is shown in figures 10 and 11. The novice goes from menu to menu picking the commands while others do not even see these menus.

PICK WHICH FUNCTIONAL AREA IS NEEDED:

CHANGE : USE THIS COMMAND TO MAKE ADDITIONS, INSERTIONS, DELETATIONS, OR REPLACEMENTS IN THE FILE.

ADMIN : USE THIS COMMAND TO SET MARGINS, CHECK SPELLING, AND LOOK AT THE TEXT OF THE FILE.

MISC : USE THIS COMMAND TO PERFORM VARIOUS MISCELLANEOUS FUNCTIONS SUCH AS THE MEMORY ADDRESS OF THE WORDS IN THE FILE.

Figure 10 Menu Showing Commands By Functional Area

SELECT ONE OF THE FOLLOWING COMMANDS TO CHANGE THE FILE:

INSERT : PLACES LETTERS OR WORDS AT THE POINT WHERE THE CURSOR IS POINTING.

COPY : TAKES TEXT FROM ANOTHER FILE AND COPIES IT INTO THIS FILE.

REPLACE : REPLACES LETTERS OR WORDS WITH OTHER LETTERS OR WORDS.

DELETE : DELETES TEXT FROM FILE.

Figure 11 Commands For CHANGE functional Area

The short menus or prompts displayed after a SEDIT primitive command offer only one level of information. Once again there is no way for anyone to get more information about the effect of the commands or to avoid the menu. If the level of experience was found at entry, appropriate messages could be displayed. Some menus may be necessary to serve as a checkpoint so that if the user decides to abort the command, the interface will bring the system back to that point. When brought to a checkpoint, different users would be shown different prompts.

The prompts in SEDIT are good in that they are very short and require short entries which reduces the load on short term memory and closure. However, the more experienced prefer to type in a series of commands without waiting for any prompt. For example the user might want to go to an old file (SEDIT OLDFILE.TXT), copy a paragraph from another file (COPY ANOTHERFILE.TXT, 50, 75, END, END), and change the word "stop" to "go" (STRING/STOP/GO/). With SEDIT each command must be typed in separately. Typing in all these command on one line would save a great deal of time. Of course every one would want each character checked as it was typed in so that it could be corrected without typing the whole line over again. It is very frustrating if the command has to be typed in over again for one mistaken character. The interface can check each letter of the command as entered so it can be retyped immediately. Another solution is to have the interface save the command, echo it back on the screen so the incorrect letter can be retyped. The minimum is to have the incorrect entry marked

in some way so the mistake is easily identified.

The QUIT command is needed on several menus throughout the interface so that the user learns know how to re-enter or leave SEDIT. QUIT is thus repetitively learned. A short reminder at the bottom of every ending prompt will do. Both the novice and the casual user need reminding on how to exit gracefully. Experienced users do not need this because they already know the command, and if not they can use a HELP command. An example of a reminder for the QUIT command at the end of the EXAMINE command is shown at figure 12.

"EXAMINE" COMMAND

THE EXAMINE COMMAND CHECKS THE SPELLING OF ALL THE WORDS IN THE TEXT. AS THE FILE IS EXAMINED AND WHEN AN INCORRECT SPELLING IS FOUND THE CURSOR IS PLACED AFTER THE WORD. THE FOLLOWING OPTIONS ARE AVAILABLE AT THIS POINT:

CORRECT SPELLING : BY TYPING IN A "C" THE SPELLING OF THE WORD IS VERIFIED AND THIS SPELLING WILL BE ACCEPTED FROM NOW ON.

<CTRL>C : CONTINUES ON CHECKING THE SPELLING OF THE TEXT. A NOTE SHOULD BE MADE OF THIS WORD TO CHANGE AND CORRECT LATER.

<ESC> : EXITS FOR THE EXAMINE COMMAND AND RETURNS TO SEDIT.

NOTE: TO LEAVE SEDIT, TYPE IN THE COMMAND "QUIT" AFTER <ESC>. THE QUIT COMMAND CAN BE ABBREVIATED TO "Q" IF DESIRED.

Figure 12 Menu For EXAMINE Command

6.5 NATURAL LANGUAGE AND UNIFORMITY

The commands in SEDIT have uniformity in that they follow the same general format. For example the command to enter SEDIT is "SEDIT filename.extension". All the commands have the same general format. The only exception is the repeat function placed before the command which may cause confusion. For uniformity throughtout SEDIT, the command should always be first.

Another lack of uniformity is in the "ESCAPE" and "CONTROL C" commands or parameters. After the INSERT command, ESCAPE deletes all inserted information while after the DELETE command ESCAPE exits without any changes to the file. There is already a command that will exit from any command, QUIT. It is a better way to exit because it takes the user back to a familiar point from which the user can easily find the way. QUIT can also save the status of the preceding command or state of the interface so a choice can be made as to whether or not a return to old point is desired or not. The QUIT menu could also let other commands be called without the loss of the old. A better way is to have the QUIT command allow exiting from one state, going to another command, performing that command, and then returning to the first. This is an especially powerful tool for the experienced user who wants to go back to some other part of the text to change another part before continuing on. An example of the benefit of such a facility occurs when the user is adding a procedure to a program with a very long INSERT command and realizes that a previous

section must be checked to see if the procedures match. With this feature, the user could exit from the INSERT command check the other procedure and then return to the exact same point and continue on.

Natural language means that commands and messages have a similar meaning in English. Since the commands must be understood by everyone, this is very difficult because of the different levels of experience. In SEDIT the command EXAMINE checks the spelling of the words in the text. Both the novice and the experienced would have a difficult time determining exactly what EXAMINE really does without reading the user's manual. Admittedly, the words in English do not lend themselves to specific computer usage, but an effort should be made to have an English word with a definition that approximates the function or command it performs. The command PAGE is very close to an English meaning in its function because it means to "page" through the text.

6.6 USER AIDS

SEDIT does not have any user aids other than terse menus and prompts. The only way a user can obtain information is to go the manual. There are several ways of providing a message for aid. One way is to let the user ask for a message about a particular command. To do this the interface has to have a HELP command. Typing in "HELP" produces a menu listing all the commands available to the user with perhaps a short definition for each. Several levels of messages, each more detailed than the last, can be

displayd by typing in question marks after HELP. Figure 13 shows the message for "HELP" and figure 14 for "HELP?". A HELP command is very good for the casual user attempting to perform some unfamiliar task. Another way for a HELP prompt to appear is when an incorrect command is entered, a HELP message is automatically displayed.

THE FOLLOWING COMMANDS ARE AVAILABLE:

INSERT	DELETE
QUIT	ADJUST
EXAMINE	COPY
MARGIN	REPLACE
JUMP	FIND
PAGE	HEX
VERIFY	SET

Figure 13 Menu For "HELP"

THE SEDIT COMMANDS ARE

INSERT : INSERT PLACES TEXT INTO THE FILE WHERE THE CURSOR IS.

DELETE : TAKES TEXT OUT OF THE FILE.

EXAMINE : CHECKS SPELLING OF WORDS.

HEX : GIVES THE MEMORY LOCATION OF THE WORDS IN THE TEXT.

JUMP : GOES DIRECTLY TO A LINE OF TEXT.

QUIT : EXITS FROM THE INTERFACE.

THE ABOVE COMMANDS MAY BE ABBREVIATED BY TYPING IN THE FIRST LETTER.

Figure 14 Menu For "HELP?"

When the command is known but more information is wanted, the command with a question mark after it displayes a menu about that command. There can be several layers of messages as in the HELP command. The more detailed the response required the more question marks typed in after the command. A series of menus produced at different levels for the REPLACE is at figures 15A and 15B.

```
REPLACE   [N]   <"S"   OR   "W">   <TARGET>
<SUBSTITUTION>
```

Figure 15A Menu For "REPLACE?"

```
THE REPLACE COMMAND REPLACES A STRING OR WORD
IN THE TEXT WITH ANOTHER STRING OR WORD.  THE
COMMAND "REPLACE" IS FOLLOWED BY:
      [NUMBER OF TIMES REPEATED] - OPTIONAL
      <STING "S" OR WORD "W">
      <TARGET STRING OR WORD TO BE REPLACED>
      <STRING  OR  WORDS  SUBSTITUTED  INTO
TARGET>
```

Figure 15B Menu For "REPLACE??"

A HELP command or query for more information should not interrupt the the present command or state; otherwise, the full usefulness of the HELP command is lost. A person would

be very hesitant to call on HELP when every time information was requested, the system returns to some entry point at the beginning of the interface. When user aids produce more work, they will not be called upon and the learning process slowed down. A definition of the QUIT command is needed in several of the HELP menus to show how to exit SEDIT for those who are totally lost. It is hoped that few will get lost with all the aids provided.

6.7 ERROR HANDLING

SEDIT does not have any error messages or recovery techniques except for the menus displayed upon entry as discussed earlier. SEDIT just ignores incorrect commands forcing another try without any information about the error. For the experienced user, this may be enough but quickly causes the novice to get lost or, in frustration, to simply type in commands or letters just to get a response. By providing a series of menus explaining the HELP command and other aids, the user could simply ask for assistance. However, it is possible for the interface to automatically provide information when needed.

If a mistake is made the machine should at least echo the command with an indication of what is wrong with the command. When making long entries, this aid is very costly in terms of software because the interface will mark each error. Difficulty increases as the number of commands on one line increase, for the interface must decide or attempt to decide, where the user is headed if the mistake appears

in the first part of the command. However, with the short commands in SEDIT multiple error correction is not a problem.

Another method for error recovery is to supply a menu listing the commands available when the error occurs, to call HELP command, or to display another user aid. Figure 16 is a menu for an error message and includes a few words about the HELP and QUIT commands.

THE MACHINE CAN NOT ACCEPT THE LAST COMMAND. THE COMMANDS THAT THE MACHINE WILL ACCEPT IN THIS STATE ARE AS FOLLOWS:

EXAMINE

MARGIN

ADJUST

HEX

SET

TO TERMINATE THE SESSION THE COMMAND IS "QUIT". TO HAVE THE ABOVE COMMANDS EXPLAINED USE THE FOLLOWING COMMAND:

<COMMAND>?

THE MORE "?" AFTER THE COMMAND THE MORE DETAIL GIVEN.

Figure 16 Example Of Error Message

A method of avoiding errors in commands is to have the screen show the commands available in the present state as SEDIT does now. The commands that the interface can accept are always shown at the top of the screen. The price in response time is well spent for the novice but would slow down all others. The sight of the line at the top all the time might create ill feelings for some. Perhaps the time could be better spent if there was a single line at the top telling how to call a HELP command or by listing those commands might be used next when done with the present command. When using the CREATE command, MARGIN, COPY, ADJUST, or SET would be at the top.

Another user aid is the creation of a command file where all the messages passing through the interface are stored. With this the user can trace through the messages to find the error. An unexpected result is best found with something like this in the system. Tracing through the system will build the user's conceptual model faster.

There is one principle that must be followed when designing any error messages. Error messages must be written in simple easy to understand English showning the user what to do next. Error messages have little value unless information contained in them leads to some sort of recover from the error, indicates what options are available, or returns to a well know point in the system.

6.8 USER CONTROL

One of the best features of SEDIT is the MACRO command permitting users to make up their own commands. A strong sense of control occurs when they can tell the system what to do according to their conceptual models. This model is the way that the user solves problems. MACRO commands permit the user to expand and refine this model and include all the interface features into it.

SEDIT does not leave any doubt as to who made the mistake in the command. It just ignores the command which produces frustration for some. It should give an indication that either an error occurred or that it is doing something. If the command takes a long time to complete, a message such as "++" could be flashed on the screen. The "++"'s do two things. First, the user knows that the command is correct. Second the user knows that the system is working on the command. For commands that need a long time to complete, a message indicating the length of the wait is needed. An example is the EXAMINE command. The interface could flash the message at figure 17 giving an estimate of the time it would take.

THE EXAMINE COMMAND GIVEN WILL TAKE APPROXIMATELY 5 MINUTES PER PAGE TO COMPLETE. DO NOT SIGN OFF UNTIL A MESSAGE IS DISPLAYED STATING THAT THE TASK HAS BEEN COMPLETED OR THERE IS A CHANCE THAT THE FILE MAY BE LOST. THE FILE BEING EXAMINED HAS 30 PAGES.

Figure 17 Message for EXAMINE Command

6.9 SUMMARY

SEDIT is an interface designed to be a tool for people with some experience. It offers a lot of power, speed, and ease of use but has not taken into account the needs of the novice and experienced users. All persons using SEDIT need to read and study the user's manual before attempting to use it. SEDIT cannot take a user through the interface by itself. It cannot help those who become lost because it does not have any error messages or recovery devices. The very experienced user is slowed down by the fact that there is only one level of commands available and no way to type in more than one command at a time.

7.0 SUMMARY

This report has discussed some of the considerations in designing an interactive interface so that it is friendly to all users. The areas of consideration that were covered were hardware/environmental issues, psychological issues, and user issues. The importance of designing interfaces for friendliness is becoming increasingly important as more and more computer users are not trained professionals but rather casual users who are interested in applying the computer to make their jobs easier. However, they consider their time very important and do not want to be forced to spend a great amount of time in learning how to use a computer.

The analysis of SEDIT shows that to incorporate user friendliness in any interface requires a significant amount of effort and an early dedication to making the interface friendly. The suggestions for improving SEDIT are simple in concept but require more software. The effort expended by the software developer will be appreciated by the user. The more friendly the interface, the greater the potential that the interface will be purchased and used.

REFERENCES

1. Barnard, P.J.; Hammond, N. V.; Morton, J.; Long, J. B. Consistency and Compatibility in Human-Computer Dialogue. International Journal of Man-Machine Studies, Vol 15, No 1, July 1981. 87-134.

2. Crysler, Earl. Some Basic Determinants of Computer Programming Productively. Communications of the ACM, Vol 21, No 6, June 1978. 472-477.

3. Cox, G. B.; Walsh, B. C. A HELP System for the User Community. Software - Practice and Experience, Vol 10, No 3, March 1980. 221-229.

4. Doherty, W. J.; Kelisky, R. P. Managing VM/CMS Systems For User Effectiveness. IBM Systems Journal, Vol 18, No 1, January 1979. 143-163.

5. DuBoulay, B.; O'Shea, T.; Monk, J. The Black Box Inside The Glass Box: Presenting Computing Concepts to Novices. International Journal of Man-Machine Studies, Vol 14, No 13, April 1981. 237-249.

6. Dzida, W.; Herda, S. User-Perceived Quality of Interactive Systems. IEEE Transactions on Software Engineering, Vol SE-4, No 4, July 1978. 270-275.

7. Gaines, Brian R. The Technology of Interaction - Dialogue Programming Rules. International Journal of Man- Machine Studies, Vol 14, No 1, January 1981. 133-150.

8. George, A.; Lui, J. W. H. The Design of a User Interface for a Sparse Matrix Package. ACM Transactions on Mathematical Software, Vol 5, No 2, June 1979. 139-162.

9. Gruenberg, Fred. Making Friends With User-Friendly. Datamation, Vol 27, No 1, January 1981. 108-112.

10. Hayes,P.; Ball, E.; Reddy, R. Breaking the Man-Machine Communication Barrier. Computer, Vol 14, No 3, March 1981. 19-30.

11. Huckle, B. A. Designing A Command Language for Inexperienced Computer Users. Proceedings of the IFIP TC 217, Working Conference on Command Languages 10-14 September 1979. 200-214.

12. Krasner, Leonard; Ullmann, Leonard. Behavior

Influence and Personality: The Social Matrix of Human Action. New York: Holt, Rinehart, and Winston, Inc; 1973.

13. Lachman, Roy ; Lachman, Janet; Butterfield, Earl C. Cognitive Psychology and Information Processing. Hillsdale, NJ: Lawrence Erlbaum Associates, 1979.

14. Lee, R. M. Conversational Aspects of Database Interactions. Proceeding of the Fourth International Conference on Very Large Databases, 13-15 September 1978. West Berlin, Germany. 392-399.

15. Lindsay, Peter.; Normann, Donald A. Human Information Processing. New York: Academic Press, 1977.

16. Ledgard, H.; Whiteside, J. A. The Natural Language of Inter- active Systems. Communications of the ACM, Vol 23, No 10 October 1980. 556-563.

17. Ledgard, Henry F.; Whiteside, John A.; Seymour, William; Singer, Andrew. An Experiment on Human Engineering Of Interactive Software. IEEE Transactions on Software Engineering, Vol SE-6, No 6, November 1980. 602-604.

18. Ling, Robert F. General Considerations on the Design of an Interactive System for Data Analysis. Communications of the ACM, Vol 23, No 3, March 1980. 147-154.

19. Lucas, Henry C. Jr. The Use of an Interactive Information Storage and Retrieval System in Medical Research. Communications of the ACM, Vol 21, No 3, March 1978. 197-205.

20. Moran, Thomas P. The Command Language Grammar: A Representation for the User Interface of Interactive Computer Systems. International Journal of Man- Machine Studies, Vol 15, No 1, July 1981. 3-50.

21. Negus, B.; Hunt, M. J.; Prentice, J. A. DIALOG: A Scheme for the Quick and Effective Production of Interactive Applications Software. Software - Practice and Experience, Vol 11, No 3, March 1981. 205-224.

22. Nelson, J. Desktop Computers Friendly Alternative In The 1980's. Data Management, Vol 18, No 9, September 1980. 29-30.

23. Palme, Jacob. How I Fought With Hardware and

Software and Succeeded. Software - Practice and Experience, Vol 8, No 1, January 1978. 77-83.

24. Palme, Jacob. A Human-Computer Interface for Non-Computer Specialists. Software - Practice and Experience, Vol 9, No 9, September 1979. 742-747.

25. Rayner, David. User Interface in Open Data Communication Networks. Proceedings of the IFIP TC 2-7; Command Language Directions, D. Beech, ed; North-Holland Publishing Co, 1980. 405-413.

26. Rayner, David. Designing User Interfaces for Friendliness. Proceedings of the IFIP TC 2-7; Command Language Directions, D. Beech, ed; North-Holland Publishing Co, 1980. 233-243.

27. Reisner, Phillis. Use of Psychological Experimentation As an Aid to Development of a Query Language. IEEE Transactions On Software Engineering, Vol SE-3, No 3, May 1977. 218-227.

28. Reisner, Phyllis. Formal Grammar and Human Factors: Design of an Interactive Graphics System. IEEE Transactions on Software Engineering, Vol SE-7, No 2, March 1981. 229-233.

29. Reisner, Phyllis. Human Factors Studies of Database Query Languages: A Surveyy and Assessment. IBM Research Report, RJ3070 (38116) Computer Science 3/2/81. IBM Research Laboratory, San Jose, California.

30. Robertson, G.; McCracken, D.; Newell, A. The ZOG Approach to Man-Machine Communication. International Journal of Man-Machine Studies, Vol 14, No 4, May 1981. 461-488.

31. Sabine, Rahlfs. Liguistic Considerations for User Interface Design. Integrated Office Systems - Burotics, Edited by Najsh Naffah. New York: North-Holland Publishing House, 1980. 189-193.

32. Schneider, B. R.; Watts, R. M. SITAR: An Interactive Text Processing System for Small Computers. Communications for the ACM, Vol 20, No 7, July 1977. 495-499.

33. Schofield, D.; Hillman, A. L.; Rodgers, J. L. MM/1, A Man- Machine Interface. Software Practice and Experience, Vol 10, No 9, September 1980. 751-763.

34. Shapiro, S. C.; Kwasny, S. C. Interactive

Consulting Via Natural Language. Communications of the ACM, Vol 18, No 8, August 1975. 459-462.

35. Shneiderman, Ben. Software Psychology. New York: Winthrop Publishers, Inc; 1980.

36. Smith, Hugh. Human-Computer Communications. Human Interaction With Computers: Edited by Smith, H. T. & Green, T. R. G.; Academic Press, New York, 1980. 5-38.

37. Sneeringer, James. User-Interface Design for Text Editors: A Case Study. Software - Practice and Experience, Vol 8, No 5, Sep-Oct 1978. 543-557.

38. Spitiolopoulos, V.; Shackel, B. Toward a Computer Interview Acceptable to the Novice User. International Journal of Man-Machine Studies, Vol 14, No 1, January 1981. 77-90.

39. Tagg, Stephen K. The User Interface of the Data Analysis Package: Some Lines of Development. International Journal of Man-Machine Studies, Vol 14, No 3, January 1981. 297-316.

40. Wilcox, T. R.; Davis, A. M. The Design and Implementation of a Table Driven, Interactive Diagnostic Porgramming System. Communications of the ACM, Vol 19, No 11, November 1976, 609-612.

41. Zloof, M. M.; de Jong, S. P. The System for Business Automation (SBA) Programming Language. Communications of the ACM, Vol 20, No 6, June 1977. 385-376.

DESIGN OF USER FRIENDLY INTERACTIVE INTERFACES

by

JOHN FRANK YORK

B. S., ILLINOIS WESLEYAN UNIVERSITY, 1966

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1982

ABSTRACT

The point where man and computer communicate is an interface. The relative ease with which one can use this interface to communicate with the computer is the user friendliness of the interface. Increasing the user friendliness of interfaces is necessary as computers are used by a wider cross-section of people. This report discusses three areas affecting the friendliness of a computer system and analyzes an interface for its friendliness.

The first area is the actual physical characteristics of the computer hardware. This includes such items as keyboard design, physical work area, and response time. Second, the psychological factors are explored. User memory, level of experience, closure, attitude and anxiety, control, and error handling are covered. Lastly, design issues for the interactive interface are broken down into three sub-areas; user issues, machine communications, and error handling, and discussed.

The last section analyzes a text editor, SEDIT, for its degree of friendliness. Ways to improve the friendliness are offered and discussed in light of the previous three sections.