THE DESIGN AND IMPLEMENTATION OF USER-ORIENTED SYSTEMS

by

Nathan Relles

Computer Sciences Technical Report #357

July 1979

The Design and Implementation of User-Oriented Systems

BY

NATHAN RELLES

A thesis submitted in partial fulfillment of requirements for the degree of

DOCTOR OF PHILOSOPHY
(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN - MADISON

1979

The Design and Implementation of User-Oriented Systems

Nathan Relles

Under the Supervision of Professor Larry E. Travis

The research described in this dissertation investigates user-oriented facilities of computer systems, that is, software features and capabilities that contribute to ease of use and learnability. These facilities are aimed at improving user performance and satisfaction by 1) simplifying communication between a user and a system, 2) improving a user's understanding of a system and its messages, 3) minimizing the likelihood of user errors, and 4) enabling users with widely differing levels of experience to use the same system.

A taxonomy of user-oriented features is developed, consisting of five categories: on-line aids, alternative modes of communication, defaults, language extensibility, and effective language design. These features benefit naive and experienced users alike, but are not provided uniformly and effectively in existing systems. A set of operating system capabilities is proposed to simplify and encourage the provision of user aids. A set of primitives realizing these capabilities was implemented in the form of a user interface that enables any system to provide user aids consistently, efficiently, and in an unobtrusive manner. The implementation includes facilities for experimenting with and evaluating alternative protocols for user aids.

An experiment was conducted to investigate the relationships among on-line aids, user performance, and ease of use. Although other categories of user-oriented facilities (communication modes, defaults, and language extension) were not empirically studied, the experiment serves as a model for such future study. The hypothesis of the experiment was that user performance and perceived ease of use would be affected by the existence of on-line aids and the manner in which those aids were provided. The user interface served both as an instrument and object of evaluation. In the first sense, the interface was used to implement the different forms of assistance under study and to monitor their use. At the same time, properties of the interface were evaluated with respect to 1) their effect on ease of use and 2) the effort required to incorporate on-line aids into a program.

Acknowledgements

I am indebted to many people for their help with this research. My advisors, Dr. Richard Venezky and Dr. Larry Travis, were an invaluable source of guidance and encouragement. I am especially grateful to Professor Venezky for his assistance after he left Madison; I could not have begun nor completed this research without him. I would also like to thank the members of my committee for their constructive criticisms and suggestions.

My association with the University of Wisconsin-Madison Academic Computing Center gave me a most favorable environment in which to work: a stable computer system, a wide range of applications and users, and a cooperative, competent staff of co-workers. Many fellow students and MACC employees contributed to my research through stimulating and challenging discussions. I cannot name them all here, but I would be remiss if I did not single out Lynne Price for all her helpful and constructive criticism.

Finally, I must express with words what words cannot convey: my boundless gratitude to my family. My parents and parents-in-law provided the support and encouragement that made this research possible. My wife, Mary, made countless sacrifices so that I could complete this work; I will be forever grateful for her unrelenting support and inspiration.

Table of Contents

1. Introduction 1

  1.1 Problem definition 6

  1.2 Scope of the study 7

2. Background 11

  2.1 HELP modes 13

  2.2 Language extensibility 20

  2.3 Alternative modes of communication 24

  2.4 Defaults and options 29

  2.5 Miscellaneous user aids 33

  2.6 Summary 35

3. User Performance and User Needs 37

  3.1 Cost-benefit tradeoffs 37

  3.2 User needs 42

    3.2.1 Learning about a system 42

    3.2.2 Correcting errors 43

    3.2.3 Communicating with a system 44

4. Supportive Facilities that Improve User Performance 47

  4.1 Goals of the Design 49

4.2 On-line aids 51

4.3 Alternative communication modes 57

4.4 Default facilities 63

4.4.1 Task-related defaults 66

4.4.2 Generalized Defaults 71

5. The Implementation of User Aids 76

5.1 A software architecture for user aids 76

5.2 Structure of the message file 83

5.3 Primitive functions of the User Interface 84

5.3.1 Initiating communication: UIOPEN 85

5.3.2 Defining aids: UIDEF and UICODE 87

5.3.3 Activating a script: UISET and UINPUT 90

5.3.4 Script parameters, references, and special characters 91

5.3.5 Initial display of a message: UISHOW 94

5.3.6 Key-directed aids: UINDEX 95

5.3.7 Monitoring system usage: UILOG and UILMSG 99

6. Evaluating the Effectiveness of User Aids 101

6.1 Design of the experiment 103

6.1.1 Hypotheses 104

6.1.2 The independent variable 106

6.1.3 The dependent variables 107

6.2 The pilot study 109

6.2.1 Objectives 109

	6.2.2 Subjects	110
	6.2.3 Procedure	111
	6.2.4 Results and Interpretation	112
	6.2.5 Summary	116
6.3	The experiment	117
	6.3.1 Hypotheses and subjects	117
	6.3.2 Procedure	119
	6.3.3 Results and Interpretation	120
6.4	Incorporating on-line aids: the programmer's view	131
6.5	Conclusions	134
7.	Summary and Future Research	137
7.1	Summary	137
7.2	Future research	139
References		143
Appendix A		153
Appendix B		160
Appendix C		165
Appendix D		169
Appendix E		170
Appendix F		173
Appendix G		174
Appendix H		177
Appendix I		179
Appendix J		181

List of Figures and Tables

Figure 2-1 Sample HELP session (Project Genie) 18

Table 3-1 Costs of using and learning to use a system 40

Figure 4-1 Use of the EXPLAIN ERROR aid 53
Figure 4-2 Use of the HELP aid 55
Figure 4-3 Use of EXPLAIN TERM 56
Figure 4-4 Use of the DOCUMENTATION aid 58
Figure 4-5 Task hierarchy 65
Figure 4-6 Illustrated use of defaults 67
Figure 4-7 Form of a PREAMBLE 72

Figure 5-1 Structure of early computing services 77
Figure 5-2 Existing structure of supportive services 78
Figure 5-3 General-purpose supportive services 79
Figure 5-4 Integration of supportive capabilities 80
Figure 5-5 The User Interface 82
Figure 5-6 Representation of two defined aids 90
Figure 5-7 Sample scripts 93
Figure 5-8 Sample session using scripts 95
Figure 5-9 Sample scripts for key-directed aids 97

Table 6-1	Analysis of variance; perceived ease of use (pilot study)	113
Table 6-2	Analysis of variance; response to question 8 (pilot study)	116
Table 6-3	Analysis of variance; response to question 8 (experiment)	121
Table 6-4	Analysis of variance; preceived ease of use (experiment)	123
Table 6-5	Analysis of variance; perceived effectiveness of on-line aids	124
Table 6-6	Analysis of variance; total cost of performing tasks	126
Table 6-7	Analysis of variance; connect time	128
Table 6-8	Analysis of variance for errors	129
Table 6-9	Summary of significant differences	136

1. Introduction

A continuing concern in the development of computer systems has been the ease with which those systems could be used. Few systems have been developed that did not lay claim to being easy to use or 'user-oriented.' The need for such systems has become apparent for a very practical (i.e., economical) reason: the cost of computing is decreasing, while the cost of people is increasing and becoming increasingly important. Martin, for example, described decreasing computer costs as follows:

> In 1955, about 100,000 program instructions could be executed for one dollar. In 1960, the same dollar bought one million, and by 1970, 100 million. In other words, the number of instructions per dollar is going up by a factor of 10 every five years. Twenty years of computer history has brought a 10,000-fold increase in power. If the increase [in computing power] continues to be exponential, as in the past, the dollar in 1980 will buy one hundred times as many computer instructions as in 1970. [MA73a, pp. 3-4]

At the same time, it is becoming more costly to train and employ people -- those who must develop and maintain software, and those who must use it. The extent to which people use a system, and their effectiveness in using it, depend on how easily they can learn and use it.

Several factors compound the problem of making systems easy to use. Among these are the widespread availability of timesharing systems, the establishment of computer networks, the decreasing cost of interactive terminals, and the plethora of systems and languages accessible from those terminals. Computing is no longer the exclusive purview of people with specialized equipment and programming skills. Today's computer system must address itself to a broadening range of users with differing needs and abilities. Similarly, a user faces a multitude of systems, each with its unique capabilities, characteristics, and protocols.

Unfortunately, there is little agreement about what constitutes a user-oriented system. Language simplicity is commonly put forth as proof of a system's user-orientation. But for every APL enthusiast, who points to that language's brevity as evidence of its simplicity, there is at least one COBOL user who prefers a more verbose means of communication. Some systems offer users innumerable options, and place these under the rubric of user-oriented features. Finally, there are systems whose ease of use is reflected in an impressive set of terminal characteristics, ranging from editing features to single-key operations. Empirical evidence is seldom provided to substantiate these claims of ease of use. Differing notions of what makes a system user-oriented remind one of Supreme Court Justice Stewart's observation on pornography: "I can't define [it], but I know it when I see it."

Lacking a precise definition, many user needs have been subordinated, if not totally neglected, at every stage of system development. In too many instances, systems have been designed with little regard for user needs. Typically, greater emphasis has been placed on the performance of the computer, as demonstrated by its utility programs, compilers, and operating system. In part, this is because cost improvements have been easier to describe and measure for computer resources than for human behavior and productivity. HELP modes, tutorial programs, on-line documentation, and similar user aids have become common on many systems, but seldom in a consistent or integrated manner. Such capabilities have usually been implemented in an ad hoc (and post hoc) fashion. Proponents of widely differing systems each claim that their system is easy to use, and a potential user has no means of evaluating those claims, except through intuition and hearsay. Ultimately, the choice of a system is usually made on the basis of other factors such as hardware costs, response times, system capacities, and the like.

Before 1974, few empirical studies of user behavior were done [GO73, GR67, KN71, RO67, SI73], and the design of systems and languages was guided by intuition and personal experience. More recently, controlled experiments have been used to investigate programmer productivity, software reliability, and the reputed advantage of one language construct or programming technique over another [GA76, GA77, GO75b, MI75, SH76a, SH76b, SH77c, WE74b, YO74]. Other studies have been directed at query languages and characteristics of database management systems [LO77a, RE75, RE77, TH75]. For the most part, these experiments have pertained to programming activities, especially those that take place outside an interactive setting. Few studies have focused on the exigencies of on-line computing or the needs of inexperienced or infrequent users [DZ78, MI74, MI77, WA74, VE77b].

The research reported here deals with the design and implementation of user-oriented systems. The first question addressed by this study is:

1) What capabilities do systems have that make them easy to use, or easier to use than otherwise comparable systems?

Chapter 2 describes features of a system commonly associated with ease of use. A taxonomy is presented that generalizes user-oriented features to five categories: on-line aids, alternative modes of communication, defaults, macro facilities, and effective language design.

A well-defined notion of user-oriented systems makes it possible to investigate the second question in this thesis:

2) What are the cost-benefit tradeoffs associated with the provision of user-oriented facilities?

Chapter 3 discusses those characteristics of man-machine interaction that must be considered in evaluating and improving ease of use. Three classes of performance-related user needs are identified: learning how to use a system, correcting errors, and communicating with a system.

The results of these investigations provide a framework for research into these two questions:

3) What features and capabilities should a system have to make it more user-oriented and how should they be provided?

4) What capabilities must an operating system have to support and encourage the implementation of user-oriented systems?

Chapter 4 proposes a set of user-oriented facilities that attend to those user needs identified in chapters 2 and 3. The capabilities are sufficiently general that they could be adapted to any class of users and within any application.

Chapter 5 describes a software user interface that could be used to provide a variety of on-line aids.

To assess the influence of user aids on ease of use, and to evaluate the effectiveness of the software user interface in particular for providing such aids, an experiment has been conducted. Chapter 6 describes the experiment and resulting conclusions. Chapter 7 compares results of this research with other studies of man-machine interaction, and suggests several areas needing further study.

1.1 Problem definition

The first thing we must do is define what is meant by a user-oriented system. Our definition must be operational in the sense that it can be used to evaluate the degree to which a system is user-oriented. It does not suffice to say that a user-oriented system is easy to use, or that it promotes a user's satisfaction with the system, though these are desirable characteristics of any system. The definition we require must have observable features that can be measured, preferably in an objective way.

In addition to humanistic concerns, the desire for user-oriented systems is motivated by the cost of human (as opposed to computer) resources. In a user-oriented system, then, total system effectiveness is sought by maximizing user effectiveness, as well as the computer system's efficiency. User effectiveness is both observable and measurable in terms of the time, effort, and resources required to use a system, and this serves our need for a practicable definition. This dissertation begins by describing user-oriented facilities on a variety of systems. It will be seen that what users and system designers commonly call user-oriented features are, in fact, those features of a system that promote the effectiveness of its users.

1.2 Scope of the study

Many factors influence user effectiveness; the ones relevant to this study are those that are features or capabilities of a computer system. Clearly, other "external" factors may influence a user's effectiveness (e.g., the user's age, the noise level of the user's environment, the printing speed of a user's terminal, and the user's motivation). Such factors are outside the scope of this study and, generally, beyond the control of system designers.

This study pertains to all types of users and systems. There is a tendency, both in the literature and in the implementation of systems, to divide user-related concerns according to classes of users, systems, or languages. It is especially common, for example, to find features aimed at so-called naive users, i.e., those with little or no prior experience with computers. Conversely, systems designed for professional programmers often have capabilities, also called user-oriented features, intended for the more experienced user. There are several drawbacks to viewing a user-oriented system as one that provides special services for users with a particular level of experience.

First, it is not clear that previous experience can be quantified, or how such measures can be used to influence a system's design. Few systems have such a small and static group of users that the previous experience of those users can be precisely defined. Depending on the nature of a system, several categories of experience affect a user's needs and abilities, for example:

programming experience,

previous experience with an interactive system,

previous experience with a similar system, and

previous experience with the system in question.

Each of these types of experience must be further stratified according to frequency and duration of use, as well as some measure of attained competency. It is therefore not surprising that very general and subjective criteria have been applied to classify users [LO77b, SH76b, WA74, VE77c]. Experience has usually been equated with programming experience or academic background. Only recently have studies begun to focus on the problem of categorizing users [FD76, MO79].

Second, it is self-defeating to provide for the needs of one group of users at the expense of another group. For example, a system designed exclusively for beginning users (with long explanatory messages, prompts for required data, and English-like commands) is likely to be unpopular and cumbersome for more experienced users. The effectiveness of a system depends on the combined effectiveness of all its users.

Third, all users will, at some time during their use of a system, forget, make mistakes, or fail to understand some aspect of the system or its output. Today's user must often deal with more than one language, increasingly sophisticated operating systems, several different editors, and a panoply of utility programs. It is a mistake to ignore the experienced user on the assumption that he or she will never need assistance from the system. A fundamental assumption underlying this research is that ease of use benefits professional programmers as well as non-programmers. Job control statements are a common meeting ground (perhaps 'battleground' is a more appropriate term) for programmers and non-programmers. In addition, many facilities originally intended for non-programmers have become essential to program development, e.g.: editing, file management, and data transfer. By considering the characteristics of users qua users, questions of effectiveness and ease of use can be addressed in a unified manner.

2. Background

The continuing desire to improve user effectiveness by making systems easy to use is reflected in many developments in computing. For example, on-line aids, structured programming, simplified interaction modes, and even the notion of a subroutine are all aimed at improving a user's effectiveness in the process of interacting with a computer. Although it is seldom stated explicitly as the goal of user-oriented features, improved user effectiveness is, in fact, the intended result of many seemingly different developments.

To analyze user-oriented features, it is necessary to distinguish between two sets of capabilities. The first set, which may be called fundamental capabilities, are those that relate to the system's purpose at the user's level of discourse. These are, generally, the tasks a user can effect, the language used to effect those tasks, and the data structures the user must understand. The fundamental components of a database management system, for example, might include

1) data records containing several fields of information,

2) a file of records arranged in some order,

3) a program that creates a file, and inserts records into that file,

4) a program that retrieves successive records from a file,

5) a program that sorts a file according to some user-specified order,

6) a program that prints the contents of a single record, and

7) an interactive language for requesting that any of the operations above be performed.

In addition, a system often includes a set of supportive capabilities, whose function is to promote the effective use of its fundamental capabilities. A recurring source of confusion and inefficiency in many systems is that user-oriented features are treated as if they were fundamental capabilities of a system. User-oriented features are supportive capabilities, and it is necessary, for purposes of evaluation as well as design , to isolate these features from a system. Most existing user-oriented facilities may be divided into four categories:

1. HELP modes

2. language extensibility

3. alternative communication modes

4. defaults

2.1 HELP modes

Many interactive systems have implemented a HELP command or HELP mode in one form or another. In a survey of 46 interactive database systems, for example, it was found that 18 of the systems provided both a HELP facility and on-line documentation, seven provided only on-line documentation, two provided only a HELP mode, and 19 provided neither [FI74]. However, a review of several different HELP facilities reveals little consensus on what constitutes 'help' for the user. The types of help vary with different categories of users, different applications, and different situations in which users require help. More noteworthy are the different ways that assistance is given on systems that are otherwise comparable in function and capability. For seemingly arbitrary reasons, help on various systems has consisted of lengthy displays of information, interactive tutorials, special-purpose programs, or no more than the phone number of a consultant. On some systems, HELP is invoked by entering special commands or depressing a single key at any time, while on other systems it is necessary to interrupt the task at hand and interact with another part of the system.

One of the earliest documented HELP facilities was for a Computer Assisted Instruction (CAI) system. The first descriptions of PLATO, a CAI system developed at the University of Illinois, included a HELP key:

> If [a student] still has difficulty answering a question, he may obtain supplementary material by depressing the 'help' button. This action causes the computer to transfer from the main sequence of slides to the beginning of the appropriate 'help' sequence. A 'help' sequence, designed to lead the student to an understanding of the main-sequence problem, is provided for every question in the main sequence. The 'help' sequence may contain a review and reformulation of previous materials pertinent to the question as well as suitable hints and suggestions. [Memory limitations of the computer] preclude the use of secondary 'help' sequences. Thus, if a student ... asks for help once more, the machine informs him that no additional aid is available. [CO62, pp. 212-213]

The PLATO system has undergone considerable development since then, and the HELP facility is no longer as limited as originally described. More than one HELP sequence can be made available, and other forms of assistance are available through more selective keys, viz., DATA, LAB, and TERM. In addition, these facilities have the following characteristics:

1) They are programmable by the system's users; that is, anyone writing a program (actually, a lesson) may specify a HELP sequence suited to his audience of users.

2) Because of the nature of the PLATO system, it is possible to make use of information about an individual user in providing help. (In practice, this is seldom done.)

3) Help is available by depressing a single key. This is more significant than might first

appear; the presence of a HELP key makes the user vividly aware of the feature and its use could not be more straightforward.

4) The user's digression from his immediate task is minimal; that is, one can obtain help without going through a complicated protocol, such as saving current information, leaving one subsystem and entering another, and finally returning to the original subsystem and restoring information.

A more limited HELP facility was designed in PLANIT (another CAI system) for users constructing lessons [BE70, pp. II.8-II.10]. When the system solicited information with an abbreviated request, entering a question mark would invoke a more explicit question. In the following example of this feature, system output is in upper case. User input is in lower case, underlined, and preceded by an asterisk (The asterisk is output by the system to indicate that information may be entered).

```
Q/M/D/P
*?

(Q)UESTION/(M)ULTIPLE-CHOICE/(D)ECISION/(P)ROGRAMMING
*d

G2.CRITERIA
*?

G2. ENTER DECISION STATEMENTS
```

On the UNIVAC 1110 time-sharing system at the University of Wisconsin, a user who makes an error in assigning a file can obtain an explanation of the error by entering 'HELP.' The following is an example (user input is again in lower case):

```
@asg,a file.
FACILITY REJECTED  400010000000

help
FAC035  FACILITY REQUEST REJECTED
FAC021  A-OPTION SPECIFIED ON @ASG;  FILE NAME NOT
        FOUND IN MASTER DIRECTORY
```

This HELP facility is currently available only for errors in file assignment, but an expanded version is under development.

In the KRONOS operating system of Control Data Corporation's CYBERNET network, a HELP command could be entered to obtain information about KRONOS commands. After entering HELP, a user could request a directory of information available, explanation (format and examples) of a particular command, or miscellaneous information (e.g., a directory of commands or a short example of a FORTRAN program). Another feature was the INFO command, which displayed general information about the CYBERNET center being accessed and its version of KRONOS [CD73].

The HELP system developed at Project Genie at the University of California, Berkeley, allows users to ask questions in standard English, and to obtain information in the same manner as they would consult a reference manual [RO70]. This form of on-line documentation is available not only for operating system functions, but also for users who want to set up HELP data bases for their own systems. A second system, called QAS, enables a user to conveniently generate a HELP facility for his own program. The system relies on a technique that recognizes significant words (keywords) of a user's question, ignoring both the order of those words and the remaining unrecognized words. Other useful features of the system include an ability to call any HELP program from another HELP program, a subroutine facility, and the saving of unanswerable questions on a file for later investigation by the system designer. Although the technique for question analysis is not complex, Roberts [RO70] reported that the system received a great deal of use, generated useful answers, and could be easily updated to allow for modifications and oversights. A sample session with the Genie HELP system is shown in Figure 2-1.

Several other systems have provided assistance in the form of on-line documentation, varying in their protocols and underlying data structures. The SPEAKEASY system developed at Argonne National Laboratory, for example, is similar in this respect to the HELP system at Project Genie, in that a large body of on-line documentation is available [CO76, CO77]. On the SPEAKEASY system, however, documentation is organized hierarchically, and is accessed by entering HELP followed by a single keyword. A keyword recognition scheme is also used in GUIDE, a program for

```
@help
TERMINATE QUESTIONS WITH A '?'.  THIS VERSION OF HELP
WILL ANSWER QUESTIONS ABOUT MATERIAL IN R-21.  TO LEARN
MORE ABOUT THE USE OF HELP, TYPE "HELP?".  (11/27/68)

?how can i read the length of a file?
USE BRS 143 WITH ATTRIBUTE NUMBER 2 TO READ THE
LENGTH OF A FILE.

?tell me about brs 143?
BRS 143:  READ STATUS.  THE CALLING SEQUENCE IS:
A:  TABLE ADDRESS OR DATA (DEPENDS ON B0).  IF A TABLE
    ADDRESS, A IS INCREMENTED TO POINT TO ONE PAST THE
    LAST WORD TRANSFERRED.
X:  THINGY NUMBER
B:  DECODES AS FOLLOWS
    BIT 0: 0 IF A HAS DATA, 1 IF A POINTS TO A TABLE
    BITS 1-11: 'TYPE' OF THINGY
    BITS 12-23: 'ATTRIBUTE'
'TYPE' IS 1 FOR A DRUM FILE, 2 FOR A JOB, FOR DETAILS
SEE PAGE 9-2 IN R-21

?what is a thingy?
A "THINGY", AS USED IN BRS143 AND 144, IS EITHER A FILE
OR A JOB.
```

Figure 2-1. Sample HELP session (Project Genie)

selecting computer science lessons on the PLATO system [PR74]. ZOG, a system developed at Carnegie-Mellon University, allows users to browse through a network of displays by successively choosing from a list of available actions [RO77]. The ZOG system allows users to mark traversed nodes and return to them at a later time. Users can also modify any display to obtain a more personalized network. The THUMB system proposed by Price [PR78] includes facilities for constructing and accessing the structural representation (*strep*) of a document. THUMB is designed to 1) organize the concepts comprising a document, 2) assist in the generation of written manuals, 3) provide on-line access to documentation, and 4) integrate written and on-line documentation. INLAT, a system under development at Bolt, Beranek, and Newman, operates on text *segments*, portions of a document that can answer a class of questions [GR76]. INLAT is intended to eventually provide on-line documentation through a Natural Language Front-End (NLF) that recognizes questions posed in English [AS77].

The feature common to all HELP facilities is an attempt to provide users with advice, instruction, or reference information when they need it. The most general and fully operational HELP facility is PLATO's; many forms of assistance can be made available simultaneously, they can be specified and tailored for any subsystem or situation, and they can store and make use of user-specific information. Unfortunately, not all systems include a CAI subsystem; those that do generally lack a clear and uniform means of integrating CAI capabilities with other portions of the system. (Factors related to the implementation of HELP facilities will be discussed in Chapter 5.) Despite the differences that exist among HELP modes, it is possible to identify several categories of assistance that users require, and that systems can provide, in an interactive setting:

1) further explanation of an error message displayed by a system,
2) further explanation of a question or request for data displayed by a system,
3) definition or description of a term pertaining to a system,
4) examples of correct input or statement sequences,
5) interactive instruction on the use of a system, or one of its commands,
6) description of a command's format,
7) reference to (or display of) relevant portions of documentation,
8) a list of available programs, commands, or capabilities, and
9) general information about a system (e.g., operating schedules, system changes, reported errors, announcements, and emergency phone numbers).

2.2 Language extensibility

To spare the user long or repetitious input and to minimize keying and spelling errors, it is common for a system to recognize abbreviations for commands. Most systems recognize single-character abbreviations, such as P for PRINT, M for MODIFY, and C for CHANGE. When more than one command begins with the same character, two or more characters may be required as an acceptable abbreviation. Some systems recognize as a valid abbreviation any initial substring of a command long enough to distinguish it from any other command. When full duplex communication is provided, the expanded command can be displayed while a user enters only its abbreviation.

In addition to system-defined abbreviations, many systems allow users to specify their own abbreviations. An information retrieval system developed for the U. S. Patent Office, for example, allows definitions of the form

let _NAME_ be _DEF_

where _NAME_ is the abbreviation or synonym, and _DEF_ is the string of characters that the system substitutes for _NAME_ when _NAME_ is entered by a user [GL72]. Where such a capability is provided, a user can define synonyms, terms that are more meaningful to him than those provided by the system, or abbreviations for command sequences that are used often without variation.

Job control languages often provide such facilities, but require some knowledge on the part of the user beyond that of his particular application. A typical example is the UNIVAC 1110 operating system, on which any number of line images may be invoked by an @ADD command. The use of this facility requires some familiarity with 1110 files, their structure, and their management.

The definition and use of abbreviations for commands (or a sequence of commands) are degenerate forms of a general macro facility. Macro processors are seldom available to all users; they are usually restricted to a single subsystem (e.g., assemblers). Some operating systems provide macro capabilities more consistently through a special purpose system that must be used as a pre-processor to any other system. In addition to one-for-one string substitution, macro processors can provide the following capabilities:

1) macro arguments: The definition may contain parameters, suitably marked; at the time of macro generation, user-specified arguments are substituted for the parameters.

2) conditional string expansion: The substitution of a definition is predicated on some set of conditions specified in the definition.

3) repeated string expansion: Part or all of a definition string may be generated several times, based on some user-specified value.

4) transfer of control; Portions of a definition may be labeled: together with conditional statements, statements in the definition can cause generation to proceed at non-adjacent portions of the definition.

5) nested macros: Definitions may define or invoke other macros.

6) macro libraries: The system allows users to store macro definitions in a system library (e.g., a file), so that they may be used in other interactive sessions.

7) optional string expansion: At the user's option, the expanded definition is displayed when it is invoked. In an interactive setting,

the user's approval may be required for the displayed expansion to take effect.

8) missing argument specification: When arguments are missing from the user's invocation of a macro in an interactive setting, the system requests the user to enter those arguments as they are referenced in the definition. (The U. S. Patent Office system mentioned above allows user-specified messages to be displayed when arguments must be entered.)

Like HELP commands, macro capabilities have existed in many forms, and no system provides all of the capabilities above in a consistent and convenient fashion. In most cases, a single rudimentary capability (e.g., command abbreviation, synonym definition, or the substitution of many lines for one) has been provided and cited as evidence of a system's ease of use. Such capabilities have usually been described and implemented as fundamental features of a particular language, hampering their uniformity throughout a family of systems. Generally, only programming languages have provided macro capabilities for argument substitution, conditional string expansion, repeated string expansion, and transfer of control.

Language extension should be provided consistently and conveniently among the different systems comprising a larger system. This will be discussed in Chapter 5. For now, we may note that macros can enable a user to redefine the language he must use, tailoring it to his own vocabulary, application, or style of interaction. Macros can also be used to minimize repetition on the part of a user. In addition to aiding users in the construction and entry of commands, macros can make commands more readable for purposes of modification. Existing macro facilities, however limited or inconsistent, are often associated with a system's ease of use.

2.3 Alternative communication modes

In an interactive setting, the dialogue between a user and a computer system requires the specification of a task to be performed, some indication of the manner in which the task is to be performed, or data to be used in performing the task. This information may be communicated in several ways. Martin, for example, has described no less than eighteen categories of man-machine dialogues, with their respective advantages and disadvantages [MA73a, pp. 12-13]. The modes of communication used by all interactive systems fall into two general categories: user-constructed commands and system-initiated requests.

The earliest interactive systems were extensions of existing batch systems, and it was therefore natural to use the same mode of communication, viz., a command language. This was not only true of job control languages designed for programmers and generally experienced users; it also became the primary means of communication provided for reservation systems, banking systems, information retrieval systems, and similar systems whose users were likely to be less experienced with computers, or whose computer usage was less frequent.

Despite the best efforts of system designers, many command languages have been found to be difficult to learn and use. This is so not only because of the many commands and options it is necessary to remember, but because of the constricting syntactic rules, naming conventions, and reserved words. For beginning or infrequent users, the complexity of a command language may preclude their continued use of a system. In situations where a computer system must be used, the continuous training of users requires a considerable investment in educational facilities, staffing, and documentation. As the number and sophistication of a system's capabilities increases, even the most experienced user will occasionally forget the exact syntax and restrictions of commands.

For these reasons, many systems have provided a different form of communication, usually called a prompt mode. In this mode, the system asks the user what tasks he wants performed or how they are to be performed. The user does not have to remember what specifications he must make, or their order, because the system asks for all necessary information. A well-designed prompt mode can often enable users to interact with a system with little or no previous training. Complemented by HELP modes of the type described above in Section 2.1, a prompt mode can perform a good part of the training function.

A variation of prompting is represented by so-called 'form fill-in' modes, which display a form on a CRT and allow the user to fill in appropriate parameters, and 'menu-selection' modes, which display a list of alternatives selectable by a user. The form-oriented approach was identified by the CODASYL End User Facility Task Group as being best suited to a broad range of users, especially those who are not data processing experts [FD76]. This means of communication, it is claimed, is most natural for users who are accustomed to filling out forms as part of their normal work.

Systems often provide a combination of prompt modes and command (language) modes. Some systems determine which specifications are better suited to each mode and alternate between those modes during interaction with the user. This is a typical approach in systems that require the user to specify his task by entering a single command, and then prompt the user for additional specifications pertinent to the task. Alternatively, a system can allow the user to enter all specifications in the form of commands with (at the user's option) related parameters. If the user omits any parameter, the system switches to a prompt mode that solicits those additional parameters. Several operating systems have offered this feature when interpreting job control statements. Finally, it is possible to leave mode selection entirely at the discretion of the user. JPLDIS, a database management system developed at Jet Propulsion Laboratories, offers this option to a user when records are being added to a database [JP75]. When adding employee records to a database, for example, a user may choose a 'free format mode,' in which records may be entered as follows:

```
NAME=JONES, SALARY=4.75, AGE=36
NAME=SMITH, SALARY=5.00, AGE=33
NAME=JOHNSON, SALARY=4.75, AGE=25
```

Alternatively, the user may select a prompt mode, in which the system prompts the user for each item. This is illustrated in the following dialogue (system output is in upper case, user input is underlined in lower case):

```
NAME:jones
SALARY:4.75
AGE:36

NAME:smith
SALARY:5.00
AGE:33
```

When several items of data must be entered, such as a command name and its qualifications or options, three alternatives can be considered:

1) fixed fields, wherein each item must occur in specified columns in an input image,
2) ordered fields, wherein items must occur in a specific order, but are separated by blanks, special characters, or reserved words,
3) labeled fields, wherein items may occur in any order, are separated by any number of blanks, and are identified by a name or other field identification.

Although different systems often require one of these formats, and claim that it provides a user-oriented (i.e., easy to use) means of interacting, there is no evidence that any one method is superior to another. In a study by Root and Sadacca [RO67], no significant differences in number of errors were observed among subjects who used the three forms of data entry described above.

There are a few instances where a single mode of communication is required by the very nature of a system. A prompting mode, for example, is the only reasonable means of communication for a drive-in automated bank teller system, whose user audience is well-defined, and whose tasks are very few and simple. A very terse command language, on the other hand, is required on a system with time constraints for message transmission and task completion, such as a reservation system. Most systems must provide for communication with a broad range of users. If command languages can be an obstacle to inexperienced or infrequent users, prompting modes can be just as annoying to the user who knows exactly what needs to be specified and wants to get about the task of making those specifications.

It seems necessary, therefore, that a system provide alternative modes of communicating commands and data to a system, so that a user may choose the mode that best suits his needs. As with other user aids, the provision of alternative communication modes has been marked by inconsistency and arbitrarily established protocols. In most systems, only a single mode is initially made available; later, other modes are implemented to cover anomalous situations.

2.4 Defaults and options

Another method used to reduce user specifications is to provide default values or conditions whenever parameters are not specified by the user. Defaults become more necessary as the number and kinds of options increase in a system. Thus, it is common to find systems whose commands may be qualified by a prodigious number of options; to make a system more 'user-oriented', defaults are provided so that a user does not have to remember or specify all available options. The values chosen as defaults are intended to satisfy the majority of a system's users, or to prevent undesired results, such as very time-consuming or expensive processing.

Few systems allow more than a single set of defaults: the system's defaults. It is seldom possible to create a set of defaults for individual users, groups of users, individual subsystems, or combinations of these. One way to provide such defaults is by a suitable redefinition of a system (and its defaults) through the use of macros. The Demand User's Monitor (DUM), developed at the University of Maryland, is one example of this approach [HA72]. A set of special-purpose programs and macros is provided for the user in the form of a pseudo job control language. A user who is told to use the resulting command language is unaware that the underlying macros and programs create a set of default specifications tailored to his needs.

A multi-level default capability was adopted in the design of LEXICO, a lexicographic text-processing system developed at the University of Wisconsin [VE77a, VE77b, VE77c]. The user's database consists of a <u>collection</u>, which is a group of texts. The user can maintain several collections, and within each one he can add and perform operations on several texts, e.g., editing, generating concordances, and classifying concordance entries under dictionary entries. The first level of defaults is at the system level. A number of text characteristics are unlikely to vary among most users, so when not otherwise specified, system defaults for those parameters apply. However, a user can specify that a different set of defaults is to be associated with a collection. This is the second default level; once specified, collection defaults are used for any operations within that collection. Finally, a set of defaults can be associated with each text in a collection. Once specified, text defaults are used for any operations on that text. Of course, a user can always override a default by explicit declaration. This capability makes it possible for a user to tailor the system to his requirements and, at the same time, minimize the number of specifications that must be entered each time a task is performed. In addition. a user need not even know about the various default levels if he is able to abide by the system defaults.

Although defaults are intended to facilitate a system's use, their provision has generally been deficient in several respects. First, the values chosen to serve as system defaults are usually arrived at in an arbitrary fashion. Users are seldom surveyed to determine which parameters, when omitted, should assume system defaults, or what those defaults should be. Computers have eminently suitable facilities for monitoring a system's use to aid in the selection of defaults, but these capabilities are seldom exploited.

Second, defaults are sometimes inconsistent among different subsystems of an operating system, or among different tasks comprising a system. Under such conditions, it is understandable that users will be reluctant to omit a parameter and trust the resulting default.

Finally, users are seldom provided with a convenient means of determining which parameters may be omitted for a given task or command, and the results of such an omission. This information is usually enmeshed in written documentation or must be determined by a complex sequence of rules. A default capability is of questionable value if a user cannot easily determine (e.g., in conjunction with appropriate on-line aids):

a) which parameters may be omitted;

b) what the result is of omitting a particular parameter (for example, does the system assume the value last specified by the user in a similar task, in the same task, or the last time it was ever specified? Or does a system default apply?); and

c) what default values are in effect at any given time, and whether or not they can be changed. Can they be changed?

2.5 Miscellaneous user aids

Several other techniques have been used to reduce a system's apparent complexity, to simplify the interaction that takes place between a user and a system, or to provide for users with varying degrees of competency. Some of these additional techniques that bear mentioning are: function keys, data-dependent aids, 'brief' modes, and multi-level (or 'layered') languages.

Function keys and buttons can be thought of as a mechanized form of macros. By reducing a user's input to a single keystroke for frequently used functions, the likelihood of a keying or spelling error can be all but eliminated. Interaction is also simplified by the ability to recognize valid requests, rather than having to recall and construct them. The same simplification is often provided by other input forms, such as light-pens, cursor positioning, and touch-sensitive screens.

Many systems allow users to obtain auxiliary information about their data. Database management systems, for example, usually provide special commands for determining the size of a database, the names of records and fields, the last time of access or update, searchable fields, displayable fields, and the time span covered by the database [MA75b].

The amount of information communicated by a system can be varied by providing 'brief' and 'verbose' modes, selectable by the user through appropriate user aids or commands. These modes determine the nature of the dialogue (e.g., the level of detail in system messages), the amount of data displayed in response to a user's request, or both. Marcus, for example, has described six such mode pairs in CONIT, a common interface for accessing bibliographic information retrieval systems [MA76]:

1) VERBOSE/TERSE: affecting the length and comprehensiveness of the dialogue,
2) INSTRUCTIONAL/SERVICE: affecting how much emphasis in the dialogue is placed on instruction as opposed to the retrieval service,
3) INTERPRETED/STRICT: determining whether a user's search terms should be extended (e.g., to include related terms),
4) ASSISTED/AUTOMATIC: determining, for example, whether any extensions should be applied automatically or with the user's intervention,
5) HIDDEN/EXPOSITORY: affecting, for example, how much the user is informed of underlying network connection procedures, and
6) VIRTUAL/TRANSPARENT: affecting a user's ability to communicate directly with a connected system in the network.

Bothersome complexity of a language is often alleviated by revealing different subsets (or 'levels') to different users, depending on their needs or levels of competency. As

Hoare has pointed out [HO73], this attempt at achieving simplicity in a language has serious pitfalls. A user who inadvertently invokes a capability which was hidden from him will not understand corresponding error messages or, even worse, may not realize the effect of his unintentional declarations. Nonetheless, the categorization of language constructs according to levels of complexity can serve as a tool in learning the complete language. As a result of experiments with SEQUEL, a database query language, Reisner concluded that the language's features should be partitioned into layers of increasing difficulty [RE77]. The experiments were also used to identify the relative difficulty of the language's basic features.

2.6 Summary

Many systems have provided supportive capabilities under the rubric of user aids or user-oriented features. These have generally been limited in scope, inconsistent among similar systems, or cumbersome to use. There is little agreement as to how any single supportive capability should be provided. The gathering of empirical evidence on which to base a more effective design has neither preceded nor resulted from the provision of user aids. Furthermore, it is often difficult to ascertain which features of a system are user aids and which are fundamental capabilities. The taxonomy of user aids suggested in this chapter is useful in making this distinction, and, as will be shown later, provides a framework for issues related to implementation and evaluation. Despite their shortcomings, user aids have been aimed at

1) reducing the amount of data communicated between a user and a system for a given task,
2) improving a user's understanding of a system,
3) reducing the amount of information that must be remembered by a user during interaction,
4) minimizing the likelihood of user errors, and
5) enabling users with widely differing levels of competency to use the same system.

The common goal of these efforts is to improve user effectiveness. The proper design and implementation of user aids requires that we define more precisely what constitutes user effectiveness; that is the subject of the next chapter.

3. User Performance and User Needs

User aids of the types described thus far do not come without a price. They require not only additional computer resources for their operation, but the resources that go into their design, implementation, and maintenance. To justify these investments, we must consider the benefits that derive from making a system more user-oriented. In this chapter, the following questions are therefore addressed:

1) What are the cost-benefit tradeoffs associated with the provision of user-oriented facilities?

2) What are the functions of a user aid from the user's standpoint? That is, what performance-related user needs can be attended to by the provision of on-line assistance, default facilities, etc.?

3.1 Cost-benefit tradeoffs

Performance evaluation has traditionally been concerned with the efficient use of computer resources. Several studies have recently given greater attention to user performance, recognizing, as stated in one such study, that

> ... there may be qualities of service which are not directly measurable in terms of cost or time to run the job. Examples of such qualities are accuracy, comprehensibility, ease of use, dependability, and the like. Though possibly not directly measurable in terms of time or cost, they may be considered through their indirect effect on either the cost or benefit side of the analysis. For example, an inaccurate system may increase costs by requiring transactions (or entire jobs) to be rerun when errors occur. Similarly, an easy to use system is more beneficial than one that is not so, because of the costs incurred as a result of user errors or delay. [AB75]

In this more global view of a man-machine system, we are interested in those advantages and benefits that are external to the computer. Several possible benefits may be readily identified. We assume, for example, that a system that is easy to use will enjoy greater use, and the implication is clear for systems whose revenue depends on the number of users and their volume of usage. The provision of on-line aids may reduce the written documentation, consulting, and educational facilities required to support a system's existence. On systems with large numbers of users, there is an advantage in the timely (and less costly) reporting of errors, system changes, and new capabilities. This is especially true where users are geographically dispersed. Finally, we must include a consideration of the user's effectiveness, that is, the amount of time that a user must spend to solve a problem or accomplish some task by interacting with a computing system.

A comprehensive analysis of total system efficiency requires that we consider all of the tasks and resources that comprise man-machine interaction. In the process of using a system to perform some task, a user will expend various resources to

1) learn how to use the system, or those aspects of the system that pertain to his task,

2) recognize, understand, and correct errors made in the course of performing the task, and

3) complete a number of steps (in conjunction with the computer) until the desired result is achieved.

These activities are present in any system, regardless of whether it is interactive or batch, and for any user: programmer or non-programmer, experienced or not.

In addition to the time spent in using and learning how to use a system, several resources are required to assist a user, namely: computer resources, other people's time (instructors and consultants), written materials, and education-related expenses such as classrooms, equipment, travel, and lodging. The most useful common measure of these resources is, of course, cost. The costs represented in Table 3-1 have heretofore not been included in performance evaluation, and merit investigation because of their increasing role in total system efficiency. We would like to investigate, for example, the cost-benefit tradeoffs that result from providing on-line aids, alternative modes of communication, or language extension capabilities.

	Learning to use system	Recognizing and correcting errors	Communicating with system in performance of task(s)
User's time	C_{UL}	C_{UE}	C_{UC}
Computer Resources	C_{CL}	C_{CE}	C_{CC}
Time spent by others to train or consult	C_{TL}	C_{TE}	
Written material	C_{WL}	C_{WE}	C_{WC}
Misc. (travel, classrooms, etc.)	C_{ML}	C_{ME}	

Table 3-1. Costs of using and learning to use a system

We shall return to the question of cost-benefit evaluation in Chapter 6. For the moment, we may observe that not all of the costs in Table 3-1 lend themselves to practical methods of measurement. Some costs are readily attainable by suitable system monitoring. We can obtain, with no appreciable disturbance to user or system, accurate measures of those activities that take place during interaction. But we cannot always determine for a given system the costs associated with consultants and training personnel, written materials, classrooms, and equipment.

These costs are generally shared by many systems and users, and are not easily apportioned on a system-by-system basis. Much of the time spent in learning about a system and diagnosing errors does not take place during interaction, and therefore cannot be easily monitored. For measures of these factors, as well as measures of a user's general ability to use a system effectively, we must rely on subjective instruments or inferential statistics.

Despite the difficulty of measuring precisely the economic benefits of a user-oriented system, it is clear that such benefits exist. The benefit we are most interested in is improved user effectiveness. When the use of a system entails long periods of use, such as in the writing and debugging of a program, the time required to complete the task may be more significant (in terms of cost) than any other single resource. When the performance of a task is brief, as in the requesting of a report or updating of a database, the user (or the ultimate recipient of the service) is interested in obtaining correct results in as short a time as possible. In either case, it is the user's performance that we are interested in improving.

3.2 User needs

User effectiveness can be defined as a measure of a user's ability to solve a problem or accomplish some task by interacting with a computing system. The efficiency with which a user can perform tasks depends largely on the time that must be spent in the three activities described above: learning, correcting errors, and communicating with the system. By identifying the needs of a user as they pertain to each of these activities, we provide a framework within which effective user aids may be designed.

3.2.1 Learning about a system

The first and most continuing need of a user is to know what tasks a system can perform and how to request that they be performed. In addition to needing an initial awareness of a system's capabilities, users will always be subject to the human frailty of forgetting, and will therefore require brief retraining. Users also need to be informed of new system capabilities and changes in existing capabilities; the frequency of modifications is a significant factor in determining the need for such awareness.

It is also desirable that a user be required to learn as little as possible about a system before beginning to interact with it. This is so because additional (perhaps more effective) learning takes place in the process of using a system, and learning will be more effective the sooner it is put into practice. New users will be less reluctant to use a system if the training initially required is minimized. This is especially true when the system will be used only once for a short period of time.

3.2.2 Correcting errors

When an error occurs, a user must be informed of the error, he must understand the nature of the error and its cause, and he must be able to correct the error easily and efficiently (e.g., with as little respecification as possible).

These requirements imply several corollary features of a system. If a user is to understand errors, error messages must be at his level of discourse; a system must prevent the display of lower-level error messages, and express error conditions in the user's terms. Error messages must be appropriate to the user-system environment. In an on-line system, for example, messages should be brief, but offer the possibility of obtaining more detailed explanation or references to documentation. Systems that attempt to correct a user's errors must inform the user of any such corrections.

The detection and correction of errors comprise a major part of man-machine interaction. A user's performance depends not only on the ability to recognize and correct errors efficiently, but on the ability to avoid those errors in the first place. This has several implications. First, the language and protocols used to communicate with the system must be designed in such a way that the likelihood of user errors is minimized, and that such errors may be easily recognized when they do occur. Secondly, when a system requires information from a user, the user must understand how that information should be entered. Finally, facilities must be provided that eliminate repetitious specification by a user, especially since repetition increases the frequency of errors for a given task.

3.2.3 Communicating with a system

A user must understand several types of information obtained from a system: error messages, warnings, messages that solicit a response, and messages of an informative nature (e.g., file status, program status, data description, and availability of facilities). All of these messages must have the characteristics described above for error messages. They must be clear and concise, and they must provide further explication when required by a user. That so many systems sacrifice intelligibility of messages for greater 'efficiency' has often been cited as an example of misdirected design priorities [GI77, KE74, KL73]. Even when transmission rates, system response times, and memory considerations are a limiting factor on message lengths, it must be recognized that repeated user errors will adversely affect a system's performance.

During interaction, a user must be able to communicate his needs to a system in an efficient manner. The time and effort required to perform tasks depends on the number of specifications that must be made, how long it takes to compose those specifications, and how much effort is required to enter the specifications. The most effective means of entering commands, options, and data depends on user-related conditions. A beginning or infrequent user needs assistance in making acceptable requests, while other users requires more concise protocols and language constructs. A system must therefore provide alternative means of communication and enable a user to select those appropriate to his needs at any given time.

One of the most pervasive needs of a user is the ability to concentrate on his immediate problem. A user must be able to formulate problems in terms that correspond, as closely as possible, to those used in his own problem domain. This characteristic of systems is sometimes referred to as transparency; the underlying programs and data structures that interpret commands, effect system actions, and provide suitable responses may (and should) be made transparent to a user. During interaction, a user must be able to enter requests with minimal digression from the vocabulary and conceptual framework appropriate to his immediate problem. A user should not need to know about a system's internal data structures and procedures in order to use a system; he should be able to alter the vocabulary of his interactive language to suit his needs, and facilities that are not directly related to the immediate task should be usable in an unobtrusive manner.

4. Supportive Facilities that Improve User Performance

The user needs identified in the previous chapter may be met, and associated efficiencies realized, by providing

1) on-line assistance,
2) alternative modes of communication,
3) default facilities,
4) language extension facilities, and
5) user-oriented language design.

This chapter proposes a user's view of the first three of these features. No single one of these capabilities or features is claimed to be sufficient for achieving significant improvements in user performance. Some require or may be more effective in the presence of another of the facilities.

While the capabilities and principles described in this chapter are applicable in any man-machine environment, they are particularly well-suited to interactive task specification systems, or their respective command languages. Warshall has described several characteristics of this class of languages [WA72]:

1) they have a broad syntax, with many rules and few common phrases,

2) there are many reserved identifiers, even when some of them are logically identical,

3) they have rigid format requirements,

4) they are usually easy to read but difficult to write, and

5) they are difficult to extend.

Such languages are usually devoid of control statements; problem descriptions are not algorithmic or self-modifying. Rather, each statement corresponds to a single task or operation, and consists of a command together with parameters that specify how the task is to be performed or on what objects it is to operate. One example of such a language is PROPHET, a system for handling information in pharmacology research [RA72]. Part of the grammar for the PROPHET language is represented below:

```
ADD {COLUMN[S]} TO  tablename  [FROM tablename]
    {  ROW[S] }

ENTER  {COLUMN[S]} {names  } OF tablename
       {  ROW[S] } {numbers}

FILLIN  tablename  FROM  tablename

{DISPLAY} {COLUMN[S]} {names  }  OF tablename
{PRINT  } {  ROW[S] } {numbers}

LIST {OTHER     }   TABLES
     {log-in-id }

SORT  tablename  BY  COLUMN[S] {names  }
                               {numbers}
```

There are many such languages, but perhaps the most common is the job control language for an operating system. The capabilities described in this chapter are therefore not intended only for the naive user, the infrequent user, or the user whose sole means of communication is a command language. These capabilities benefit experienced users and programmers who must edit, compile, and test their programs, and manage attending resources, through a job control language.

Many non-interactive systems (those that do not require user intervention, or whose processing and output precludes their use in an interactive setting) can and should provide interactive interfaces. A concordance-generating system, for example, may provide an interactive specification system for declaring concordance parameters, and check for the validity and consistency of those parameters. This approach was taken in the design of LEXICO [VE77b], and together with a set of on-line aids demonstrated the feasibility and usefulness of providing an interactive specification system for a set of non-interactive tasks.

4.1 Goals of the design

Before describing in detail the user-oriented facilities proposed in this chapter, it is useful to recall the performance-related user needs that those facilities are intended to meet:

> reducing the amount of time that must be spent learning how to use a system, prior to its initial use;
>
> reducing the amount of information that must be remembered by a user during interaction; e.g., by enabling a user to learn during interaction what tasks may be performed and how they may be requested;
>
> minimizing the likelihood of user errors;
>
> making messages from the system understandable by a user;
>
> minimizing the amount of information that a user must communicate to a system to effect each task;
>
> minimizing the amount of information communicated by a system to a user;
>
> facilitating the use of a system by an inexperienced user, while not sacrificing the efficiency of more experienced users;
>
> facilitating the use of a system after long periods of inactivity;
>
> enabling a user to concentrate on the task or problem at hand;
>
> enabling a user to interact in a manner that suits his subject matter or stylistic preferences.

To be used effectively, user aids must themselves exhibit these same features. They must be easy to use and they must not interfere with the user's main task. Furthermore, they must be consistent; that is, they must be sufficiently general that their use can be the same among the different subsystems that comprise a larger system. Finally, a user must be able to learn easily what user aids exist and how they may be used.

4.2 On-line aids

In an interactive environment, many user needs are met by the on-line aids described in this section. Each capability is represented as a command which, when entered by a user, provides the indicated information. One important characteristic of the aids described in this section is that they are <u>unobtrusive</u>; that is, they are invocable in a simple and consistent manner that does not interrupt the immediate task. The aids described in this section are not intended to comprise an exhaustive list; other types of aids will be described in chapter 5.

While a single key is a very convenient means of invoking a user aid, few terminals have any function keys at all, let alone those described below. Furthermore, the cost of providing more sophisticated terminal hardware is not always justifiable for on-line aids; this depends on their expected frequency of use for any given system and group of users. On most systems, a special character in combination with a meaningful abbreviation might serve the same function as a key, to accommodate any type of terminal. In the descriptions that follow, boldface letters are used to represent user aids, regardless of whether they are provided in the form of special commands, function keys, light-pen input, or a touch-sensitive screen.

EXPLAIN ERROR

When entered after the display of an error message, this aid provides successively more detailed explanations of the error, including its possible causes, corrective actions, examples of correct input, or references to documentation. Use of this on-line aid is illustrated in the hypothetical interaction in Figure 4-1. Note that the user may, at any time, continue the task that was being performed at the time the error was made.

EXPLAIN QUESTION

When entered after the system solicits information by way of a question or prompt, this aid provides successively more detailed descriptions of valid user responses, including (for example) the format of the input, current default values, default conventions, examples of valid responses, and references to documentation. A question is often implied in the display of a single prompt character,

```
>sort master into author by author-name
NON-EXISTENT FILE: AUTHOR

>EXPLAIN ERROR

THE OUTPUT FILE OF A SORT MUST BE CREATED
PRIOR TO ITS REFERENCE IN A SORT COMMAND.

>EXPLAIN ERROR

TO CREATE A FILE CALLED 'AUTHOR'
ENTER ONE OF THE FOLLOWING COMMANDS:
    CREATE (TEMPORARY) AUTHOR
    CREATE (PERMANENT) AUTHOR
NOTE: THE NAME OF A PERMANENT FILE MAY
CONTAIN READ/WRITE KEYS.

>create (temporary) author
TEMPORARY FILE 'AUTHOR' CREATED.

>sort master into author by author-name
```

Figure 4-1. Use of the EXPLAIN ERROR aid

to avoid long questions that become bothersome after frequent use. The existence of this aid makes it possible for prompts to be terse without sacrificing the needs of the less frequent user.

EXAMPLE

When entered after the display of a system question or prompt, this aid provides examples of valid user responses or commands; when entered after the display of an error message, examples of correct responses are given.

The need for several independent user aids is demonstrated by the aids described thus far. Many systems have a single aid or command to serve several functions; but because it is possible to have both a question and an error message concurrently displayed, a user must be able to select an explanation for either. There are many instances when the user must determine what type of explication is required. The separation of aids makes it possible to obtain further information in the most concise and unobtrusive manner. It is true that a single HELP command could provide, in some pre-determined order, more detailed error messages, prompts, examples, and definitions. But a user will be reluctant to use such a facility if a lengthy protocol or sequence of displays is required. On-line aids should be function-specific; that is, a user should be able to choose from several different aids, depending on the type of assistance desired.

HELP

When entered at any time, this aid provides a more detailed explanation of an error or question. In the absence of an error condition, system prompt, or any other condition that unambiguously identifies what kind of help the user is seeking, a message is displayed that describes all other on-line aids available to the user. The HELP aid is illustrated in Figure 4-2.

```
>HELP
YOU MAY ENTER ONE OF THE FOLLOWING:
EXPLAIN ERROR      FOR FURTHER EXPLANATION OF AN ERROR
EXPLAIN QUESTION   FOR FURTHER EXPLANATION OF A QUESTION
EXAMPLE            FOR AN EXAMPLE OF CORRECT INPUT
MENU               FOR A LIST OF VALID COMMANDS
```

Figure 4-2. Use of the HELP aid

EXPLAIN TERM

When entered together with a single term, this aid provides a definition of the specified term with respect to the system being used. The definition of a term may also include its relationship to other terms and commands, and references to documentation. Figure 4-3 illustrates use of the EXPLAIN TERM aid. Aids similar to EXPLAIN TERM can include descriptions of a command's format (e.g., EXPLAIN FORMAT command, EXPLAIN OPTION option, or EXPLAIN OPTIONS command).

```
>find author=smith
 AUTHOR FIELD NOT INVERTED

>EXPLAIN ERROR
 ONLY INVERTED FIELDS MAY BE ACCESSED
 WITHIN THE FIND COMMAND.

>EXPLAIN TERM inverted
 A FIELD IS SAID TO BE INVERTED (OR INDEXED)
 WHEN ALL RECORDS CONTAINING THAT FIELD HAVE
 BEEN PROCESSED BY THE INDEX COMMAND.

 FOR LARGE AMOUNTS OF DATA, USE DEFINDEX.

 SEE ALSO SECTION 3.6 OF THE REFERENCE MANUAL.

>EXPLAIN TERM defindex

 DEFINDEX MAY BE USED TO DEFER INDEXING UNTIL A
 LATER TIME, RATHER THAN PROCESSING ALL RECORDS
 IMMEDIATELY.

 RELATED TERMS: INDEX, CONCORD, DEFINDEX, FIND
```

Figure 4-3. Use of EXPLAIN TERM

MENU

This aid, which may be entered at any point, provides successive displays of all commands that may be entered at that time. Many systems have command languages wherein a command (or selection of a task) may be followed by a limited set of other commands and specifications. Such systems are particularly well-suited to a MENU facility, which may serve as a reminder of what specifications are valid or required for a given task, and how those specifications may be made (i.e., the format of valid commands). Furthermore, the MENU facility may be used in conjunction with other on-line aids (EXPLAIN TERM, EXAMPLE, and DOCUMENTATION) to learn *ab initio* the capabilities of a system.

DOCUMENTATION

When entered together with an identification of a portion of documentation (e.g., chapter, section, subsection, or page number) this aid displays one of the following:

a) a description of the desired portion of documentation,

b) a partial table of contents of the desired documentation, or

c) the section of documentation

The use of the DOCUMENTATION aid is illustrated in the hypothetical interaction in Figure 4-4.

4.3 Alternative communication modes

We are concerned with systems whose use may be represented as a set of tasks, T, a set of parameters, P, and a sequence of user requests, R. Each task, Ti, may have several parameters required for its operation, and any parameter in P may be required by more than one task. Each

```
>print,cjr
INVALID OPTION(S).

>EXPLAIN ERROR

THE FOLLOWING OPTIONS ARE NOT VALID: R
SEE SECTION 19.3


>DOCUMENTATION 19.3
19.3 PRINT OPTIONS
     19.3.1 C: CONTINUOUS FORMS
     19.3.2 I: IMMEDIATE START
     19.3.3 J: JUSTIFIED FORMAT
     19.3.4 M: LEFT MARGIN NUMBERING
     19.3.5 N: RIGHT MARGIN NUMBERING
     19.3.6 P: PERIPHERAL PRINT
     19.3.7 X: IGNORE ERRORS

>DOCUMENTATION 19.3.5
THE LINE NUMBER OF EACH FORMATTED LINE IS PRINTED
IN THE RIGHT MARGIN, PRECEDING THE LINE, AS
DISCUSSED IN THE PRECEDING SECTION.

>print,cjn
```

Figure 4-4. Use of the DOCUMENTATION aid

task request, Ri, must identify a task and assign values to its corresponding parameters. We shall say that a parameter, Pi, has a corresponding name, Ni. Each Ti is the task identifier (a command or its abbreviation), and each Pi is a task-related qualifier, that is, any user-specified option or parameter that designates how the task is to be performed or on what objects the task is to be performed. This is depicted in the following diagram:

(Tasks and their parameters)

T_1 $(P_{i1}$ P_{i2} ...)

(User requests)

R_1

T_2 $(P_{j1}$ P_{j2} ...)

R_2

.
.
.

T_m $(P_{k1}$ P_{k2} ...)

(Parameters:) P_1 P_2 P_3 ... P_n

For example, a sorting task may require qualifiers that specify

1) the file containing records to be sorted,
2) those parts of a record to be used as sort keys,
3) whether the sort is to be ascending or descending,
4) whether the sorted records are to be printed, and
5) a file in which to store the sorted records.

A task request is constructed by entering the task identifier followed by qualifiers that may be specified in one of three ways. First, each parameter may be specified in an order prescribed by the system. If the system-prescribed order of parameters for a given task request is

$$P_{i_1} \quad P_{i_2} \quad P_{i_3} \quad \ldots \quad P_{i_m}$$

then the task may be requested by entering

$$T_i \quad V_{i_1} \quad V_{i_2} \quad \ldots \quad V_{i_m}$$

where Ti is a command name that identifies the desired task, and each Vi is the user-specified value assigned to the corresponding parameter, Pi. For purposes of illustration, we will assume that commands are preceded by a slash (/) and followed by a blank; qualifiers are separated by commas, and a semicolon may be used to continue a task request on more than one line. If, in our example of a sorting task, the order of required parameters is the same as that given above (items 1 through 5), the following is an example of a valid sort request.

```
/SORT MASTERFILE,AUTHOR,ASCENDING,NO,AUTHFILE
```

This command requests a sort to be performed on the file called MASTERFILE; the AUTHOR field is used as a sort key, the sort is to be performed in ASCENDING order, NO printed output is desired, and the sorted records are to be stored in a file called AUTHFILE.

Secondly, parameters may be specified in any order, and are identified by their respective parameter names. This method of specifying parameters is commonly called keyword parameter specification. Task requests are of the form

$$T_i \quad N_{j_1}=V_{j_1} \quad N_{j_2}=V_{j_2} \quad \ldots$$

where each N is the name of a parameter and V is the value assigned to that parameter. In the following examples of a sort task request, keywords (parameter names) are represented by boldface characters:

```
/SORT INFILE=MASTERFILE,KEY=AUTHOR,ORDER=ASCENDING,;
   OUTFILE=SORTAUTH,PRINT=NO

/SORT KEY=AUTHOR,ORDER=ASCENDING,PRINT=YES,;
   INFILE=MAST,OUTFILE=AUTHFILE
```

Finally, a parameter may be omitted from a task request, in which case the system prompts the user by displaying the parameter name, Ni, and requiring the user to enter a corresponding value. This method of establishing parameter values, which we shall call prompting, may be used for ordered parameter specification or keyword parameter specification. In the following examples, user input is underlined and is in lower case; system messages are in upper case.

```
/sort master,author
ORDER? ascending
PRINT? no
OUTFILE? authfile


/sort infile=master,key=author,order=ascending,;
   outfile=xyz
PRINT? no
```

These three forms of parameter specification are mutually unambiguous; a user may select any one of the forms to request a single task. It is unnecessary to provide different modes, wherein a user must enter each request in the same form until a different mode is selected. A user may, therefore, request each task in a manner that suits his level of experience, frequency of use, or stylistic preference. For example, the novice user, as well as the experienced user who is performing an infrequently requested task, may enter only the name of the task, and be prompted for all required parameters.

In addition to the convenience afforded by providing all three methods simultaneously, it is possible for a user to learn, in the process of interacting with a system, what parameters are required for each task, what their names are, and how commands may be constructed more efficiently. The most that a user must know is the name of each task, and even this may be determined if other user aids (e.g., the MENU aid described above) are available.

If omission of a parameter necessarily and always results in a prompt, it would seem that each time a task is requested, all of its associated parameters must be specified by the user, in one form or another. For most systems, this is an unacceptable requirement; a system's generality often depends on having many options associated with each task. It is clear that defaults must be provided for those parameters that a user does not change frequently. The mechanism proposed in the next section provides a general default facility without sacrificing benefits of the prompt mode just described.

4.4 Default facilities

Defaults are intended to reduce the number of specifications a user must enter to perform a task. They are usually provided in only two ways: by assigning predetermined values (system defaults) to unspecified parameters, or by allowing a user to choose default values to be associated with the tasks and parameters that he uses. A complete and effective default facility must provide more general default structures, and must be easy to use. Furthermore, it must enable a user to determine in a simple and consistent manner what defaults are in effect at any time, when they may be changed, and how they may be changed.

We represent a user task request as

$$T_i \quad Q_1 \quad Q_2 \quad Q_3 \quad \ldots \quad Q_{n_i}$$

where Ti is the task identifier (a command) and each Qi is a task-related qualifier that associates a value with one of the parameters required by the task. Henceforth, we will assume that any of the communication modes described in the preceding section may be used to specify the qualifiers for a task. We would like to associate a default value, Vi, with any qualifier that is omitted. But for any omitted qualifier. a default could be 1) determined by the system, 2) dependent on the task, 3) associated with the user, or 4) associated with other (declared) qualifiers. For example, let us say that a sort command requires four qualifiers: the name of the file to be sorted, an output file, a sort key, and the ordering to be used (ascending or descending). If the ordering qualifier is omitted from a sort command, we could assign

1) a system-wide default that is always used whenever ordering is not specified,

2) a default associated with the sort task only,

3) a default specified by the user, to be used whenever he omits an ordering specification from any command, or

4) a default associated with the specified file.

$$\begin{array}{lllll}
T & Q_1 & Q_2 & \ldots & Q_n \\
 & & & & \\
\quad t_1 & q_{i1} & q_{i2} & \ldots & \\
\quad t_2 & q_{j1} & q_{j2} & \ldots & \\
\quad \vdots & & & & \\
\quad t_m & q_{k1} & q_{k2} & \ldots &
\end{array}$$

Figure 4-5. Task hierarchy

A more general treatment of defaults, then, requires that we consider the contexts in which tasks are requested. A task is requested in the context of some other task, as represented in Figure 4-5. Each t represents a sub-task (or command) of the parent task, T, and may likewise have its own sub-tasks and data. This task hierarchy is reflected in many systems, most notably in operating systems, wherein the outermost task is a job or run; within a run, commands may be entered to initiate interaction with any one of several systems, each of which may in turn have its own set of commands.

4.4.1 Task-related defaults

The first type of default that may be provided is one that is associated with all tasks of a currently active parent task. By declaring a qualifier outside of any sub-command, a default is established for that qualifier for the duration of the parent task. If a task, T, is active and any of its sub-tasks, t, may be requested, the statement

$$N_i = V$$

establishes the value V as the default for parameter Pi (whose name is Ni). This has the effect of saying: "so long as task T is active, whenever a command requiring parameter Pi is entered without specifying Pi, use V for that parameter." The parameter name 'COMMAND' may be used to establish a default command.

Figure 4-6 represents an interactive session with a hypothetical text editor, many of whose commands require a parameter specifying how many lines are to be affected by the command. In statement 6, a default is established for this parameter, called LINES. In all subsequent commands in which this parameter is required but omitted, the specified

```
        /edit docfile.preface
        EDIT VERSION 4.1,  15 MAR 77,  10:45:30

  (1)   >gotoline 5
        were administered to thirtty subjects.

  (2)   >change old=thirtty,new=thirty,lines=1

  (3)   >find pattern=subjects,lines=30
        LINES: 8, 12, 25

  (4)   >gotoline 8
        whether all subjects were adequately covered by

  (5)   >change old=subjects,new=topics,lines=1

  (6)   >lines=1

  (7)   >command=gotoline

  (8)   >12
        because subjects were given time to prepare for

  (9)   >25
        additional subjects were tested.
```

Figure 4-6. Illustrated use of defaults

value (1) is used. In like manner, a default may be changed any number of times within a parent task.

In statement 7, a default command is established, so that whenever a command is not entered, the default command is used. In the example, the user chooses to establish GOTOLINE as the default command, presumably because this command is entered frequently. This is illustrated in statements 8 and 9, where the command name is omitted.

Several additional types of default declarations may be made in an interactive setting. The statement

Ni

(that is, the name of a parameter) may be entered to determine what the current default value, Vi, is for any parameter, Pi. The statement

Ni = ?

may be entered to indicate that any existing default for parameter Pi should be rescinded. In all subsequent commands where Pi is omitted, the system should prompt for it, as was done before a default was established. The statement

Ni = ?userprompt

may be entered to indicate that whenever Pi is required in a command but omitted, the user-constructed prompt, userprompt, should be displayed, and the value entered by the user at that time should be used for Pi. Finally, a user must be allowed to choose when the system should indicate that a default has been applied. Each time a default is established, the system asks whether the specified name-value pair should be displayed whenever the corresponding parameter is omitted and a default applied.

The default capabilities described thus far, together with a command interpreter that recognizes both keyword parameter specification and ordered parameter specification, enable a user to construct his own protocols for commands.

For example, the WISE bibliographic retrieval system at the University of Wisconsin includes a LIST command, whose function is to list citations from a subset of the database. The form of the LIST command is

```
/LIST s/n/xx-yy/k-h
```

where *s* is a set number, *n* is the maximum number of citations to be listed, *xx* and *yy* identify a range of years, and *k* and *h* identify a range of citation numbers. Any of the parameters *n*, *xx*, *yy*, *k*, or *h* may be omitted, in which case a system default is used. An additional command may be entered to specify other LIST-related parameters; its form is

```
/OPTIONS a/t/p
```

where *a* is YES or NO, depending on whether citations are to be listed in their entirety, *t* is YES or NO, depending on whether the listing is to be produced on the terminal, and *p* identifies a remote printer if the user chooses to have citations printed there.

The assumption implicit in this design is that the LIST parameters *s*, *n*, *xx*, *yy*, *k*, and *h* are different each time the LIST command is entered, while the parameters of the OPTIONS command are less likely to vary during interaction. In practice this is not always true; furthermore, new and infrequent users have had considerable difficulty remembering which parameters are associated with which command, in what order they must be specified, when they must be changed, and what system defaults apply when a parameter is omitted.

A more adaptable design would be afforded by a command of the form

```
/LIST set,maxcit,loyr,hiyr,locit,hicit,ai,terminal,printer
```

where _set_ is a set number, _numcit_ is the maximum number of citations to be listed, _lowyr_ and _hiyr_ specify a range of years, _locit_ and _hicit_ specify a range of citation numbers, _ai_ is YES or NO, indicating whether so-called 'additional information' is to be listed, _terminal_ is YES or NO, indicating whether a terminal listing is to be produced, and _printer_ identifies a remote printer to which the listing should be directed. Alternatively, parameters could be entered by keyword parameter specification or prompting, as described in the previous section.

While this may appear to be a formidable set of parameters, the default capabilities described above would enable a user to construct his own protocol and defaults for the LIST command. He may choose those parameters that are required in a command, those that are to be prompted for, and those that may be omitted (and their defaults). For example, the declarations

```
MAXCIT=ALL
AI=YES
TERMINAL=YES
LOCIT=0
HICIT=9999999
```

```
LOYR=?LOW YEAR:
HIYR=?HIGH YEAR:
```

would enable a user to enter all subsequent LIST commands specifying only a set number and an indication as to whether additional information should be listed. All other parameters would take on the specified defaults if omitted, and in the case of the range-of-years parameters (LOYR and HIYR) cause the indicated prompts to be displayed instead of the system prompts.

4.4.2 Generalized defaults

A single level of defaults will soon prove inadequate to the user who must declare those defaults each time a task is initiated. We will therefore allow a user to establish a sequence of declarations, called a preamble, with any task. Whenever the indicated task is initiated, the associated task preamble will take effect, as if the user had entered its constituent declarations. The sequence shown in Figure 4-7 declares statements S1 through Sn as comprising the preamble for task T, where each Si is either a valid sub-task of T or a default declaration for all sub-tasks of T. Whenever the task T is initiated, statements S1 through Sn are applied, as if the user had entered them immediately after requesting T.

PREAMBLE (T)

S_1

S_2

.
.
.

S_n

END

Figure 4-7. Form of a PREAMBLE

As each statement of a preamble is applied on task initiation, it is displayed to remind the user that a default has been established for the task. To prevent this display, a statement may be enclosed in parentheses in the preamble declaration. A statement enclosed in brackets requires the user's confirmation to take effect (e.g., the statement is displayed and the user is asked to enter 'Y' or 'YES' to accept that default). These features of a preamble make it possible for a user (or a user's intermediary) to define the defaults and protocols that are best suited to his use of a task.

In addition to task-related preambles, we must allow defaults to be associated with the objects on which a task operates. For example, in a text editing system that operates on a file, a user may need to establish different file-related defaults (preambles) for such parameters as page width, tab settings, and word delimiters. In general, we represent a task and its required parameters as

$$T_i \quad P_1, P_2, \ldots \quad P_{n_i}$$

The general form of a preamble, then, is

$$\text{PREAMBLE } (T_i , \; N_{j_1} = V_{j_1} , \; N_{j_2} = V_{j_2} \quad \ldots)$$

$$S_1$$

$$S_2$$

$$\vdots$$

$$S_k$$

END

As with the degenerate form described earlier for task-related preambles, a statement Si may be enclosed in parentheses to inhibit its display on task initiation. The effect of this more general preamble declaration is to apply the specified defaults whenever task Ti is initiated with the specified qualifications. The preamble for a user-task-qualifiers combination may be declared either by the user himself or by an intermediary whose function is to construct a suitable set of defaults and protocols for users.

Let us say, for example, that a text editing system operates on portions of a file, called segments. Editing is initiated by a statement of the form

```
/EDIT FILE=filename,SEG=segmentname
```

and among the parameters that may be specified within an EDIT task are:

PGWDTH	(page width)
TABST	(tab settings)
WDELIM	(word delimiters)

To establish page width and word delimiter defaults for all segments within a file called 'DOCUMENT', and a default for the tab settings within a segment of the file called 'CHAPTER3,' the following PREAMBLE declarations may be entered:

```
PREAMBLE (EDIT,FILE=DOCUMENT)
PGWDTH=85
WDELIM=BLANK,COMMA,HYPHEN
END

PREAMBLE (EDIT,FILE=DOCUMENT,SEG=CHAPTER3)
TABST=6,11,25
END
```

This generalized means of establishing defaults has one apparent drawback: the default capabilities are somewhat complex, and are not readily usable by the novice or infrequent user. However, we must keep in mind that defaults are generally provided for two situations. First, defaults are intended for the frequent user who would otherwise repeat task requests with the same parameters. For such a user, the preamble facility described above is reasonably straightforward and within his grasp. Second, defaults are often intended for the less experienced user of a system, and allow such a user to interact with the system by entering a minimum of specifications. For such a situation, preambles are intended to be established and maintained by an experienced intermediary acting on the user's behalf.

Similar default structures can be created and used through a suitable definition of macros. However, the use of preambles offers three significant advantages over macro definition and invocation. First, they may be used without creating and remembering the names of individual macros. Second, they are applied automatically on task initiation, without requiring explicit invocation by the user. Third, by identifying a task with which defaults are to be associated, the user may, in some cases, be informed of errors or inconsistencies when the defaults are specified, rather than at a later time when they are applied.

5. The Implementation of User Aids

If we want systems to include user aids similar to those described in the previous chapter, we must make it easy for designers and programmers to implement them. We would also like to ensure that their implementation is efficient and that a high degree of consistency is maintained across different systems. These implementation goals may be realized by providing a set of primitive and suitably general capabilities in the form of a front-end facility. The absence of such a facility is one reason that user aids have previously been implemented in an ad hoc fashion, if at all. To see why this is so, it is instructive to look at the development of interactive systems and the concomitant development of user aids.

5.1 A software architecture for user aids

The first computing systems provided the simplest form of user services. Each user had only one system or language at his disposal: assembly language. Aside from rudimentary start-up facilities (e.g., bootstrapping, switch setting, and tape mounting), a system could be viewed by user and system implementer alike as a single-service system. The simplicity of this structure accounts for the popularity of some 'single language' systems to this day. As the number of services offered by a computer increased, a monitor (operating system) became necessary. The resulting software architecture, represented in Figure 5-1, predominates in most contemporary systems.

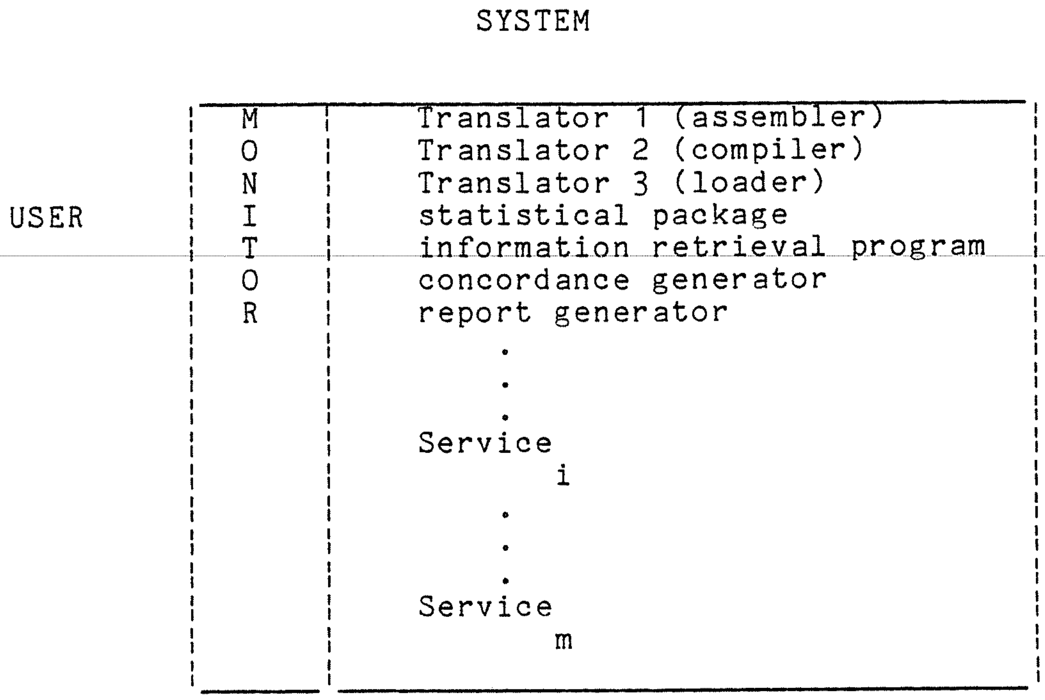

Figure 5-1. Structure of early computing services

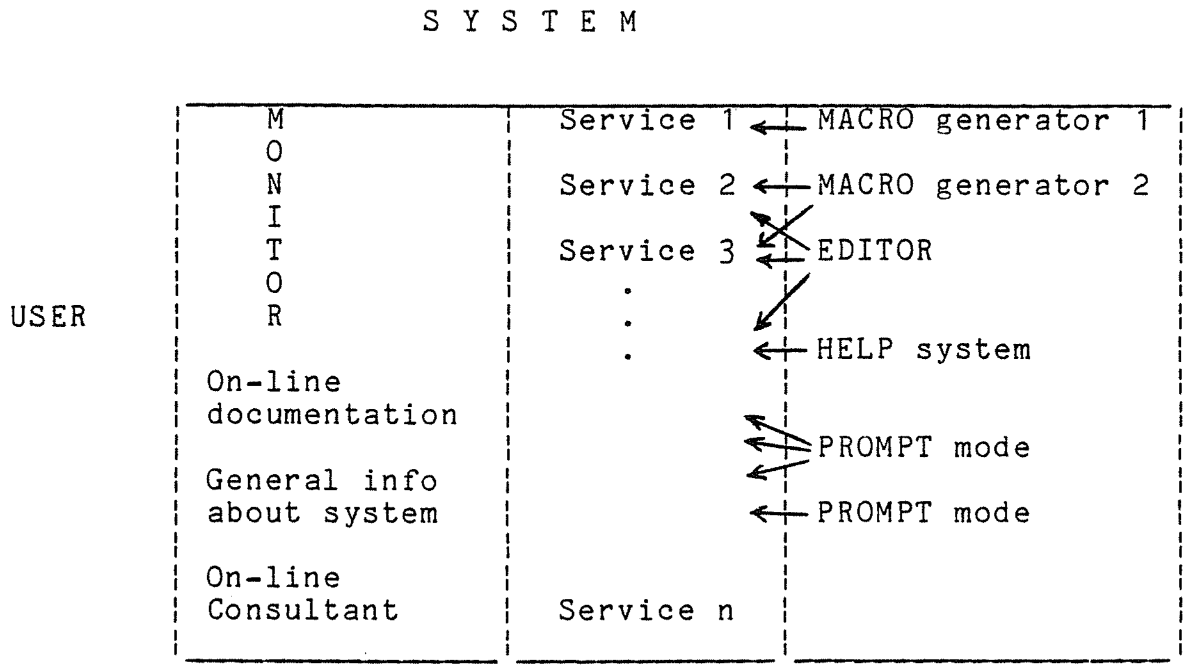

Figure 5-2. Existing structure of supportive services

Most user-oriented aids and features have been implemented within individual services of a system, preventing general availability and consistency throughout a system. Many systems reflect this patchwork of supportive services, represented in Figure 5-2. The duplication in such configurations has obvious economic drawbacks in designing and maintaining user aids among several systems. The benefits derived by users are further offset by the difficulty of using and learning about different services whose aiding features are inconsistent, redundant, or totally lacking.

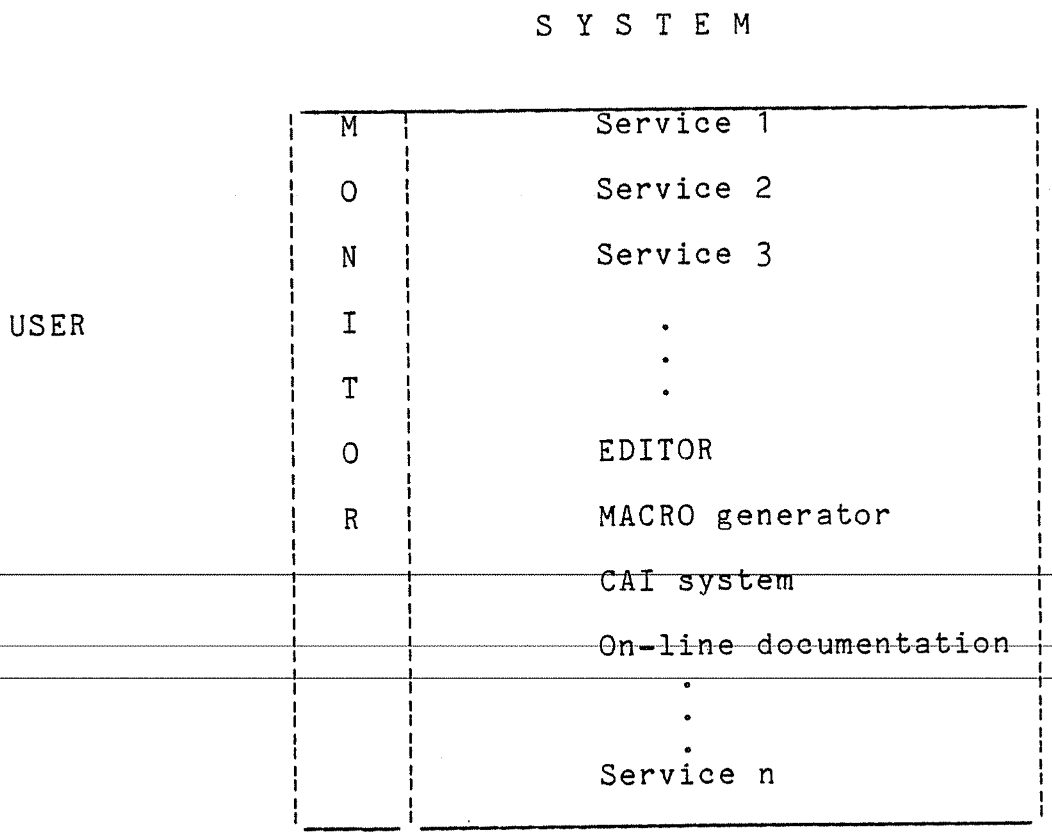

Figure 5-3. General purpose supportive services

Another approach taken in many systems is to provide user aids as separate supporting services available through the operating system, as illustrated in Figure 5-3. Unfortunately, some user-oriented features do not lend themselves to such a structure, e.g., prompt modes and defaults. A more serious drawback is that in order to invoke a particular user aid, many systems require that a user interrupt his task, return to the operating system, invoke the supporting service, return to the operating system, and then resume the original task. In addition to the inconvenience posed by this procedure, it often requires an understanding of the operating system, its data structures, and its job control language. Under the UNIX operating system, a running program can invoke another process. This mechanism allows a user to access a separate on-line documentation program, for instance, without interrupting a current task. However, this can be done only if the programmer has taken appropriate steps to provide for such interruption; in addition, the code required to do this is cumbersome.

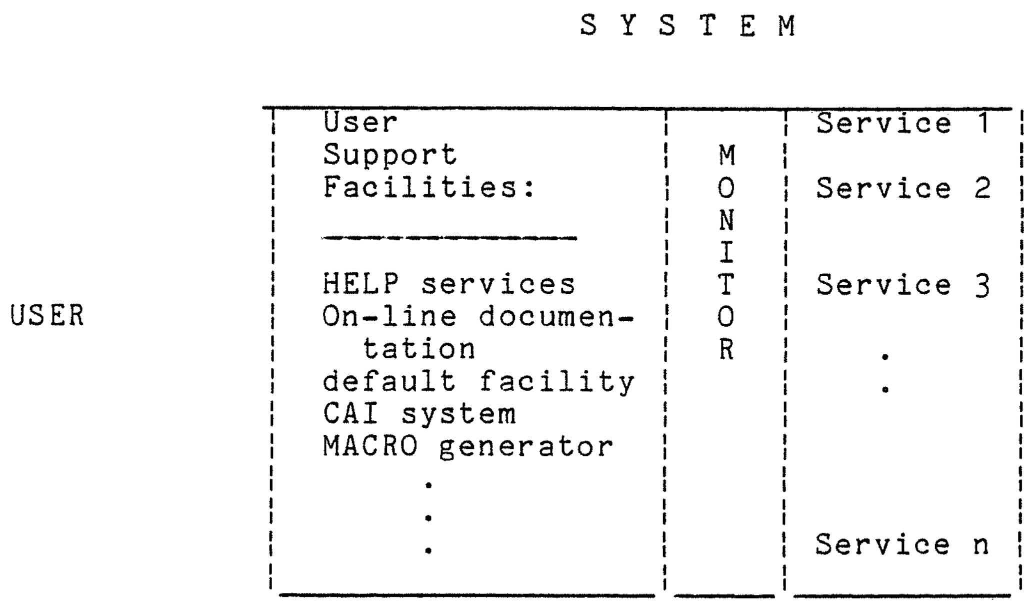

Figure 5-4. Integration of supportive capabilities

User aids must be consistent, efficient, and convenient to use both from the standpoint of system implementation and from the user's conceptualization of those aids. The software architecture that best lends itself to such implementation (by the programmer) and conceptualization (by the user) is depicted in Figure 5-4. The collection of user aids may be thought of as a front-end facility, whose function is to provide various types of assistance *at any time* and *with minimal digression* from a user's main task. This is not unlike having a human consultant at one's side while interacting with a system. By integrating user aids in this fashion, we are able to meet the user needs identified in Chapter 3 consistently and efficiently.

Beyond our concern for efficiency, this uniform approach to user aids can provide a degree of 'user-independence.' If the nature of supportive facilities were more similar among different systems, users could have a common language of interaction, namely: that set of aids, instructions and facilities that allows a user to learn on any system what can be done and how. A common interface of user aids would make interactive systems more portable from one user to another.

The remaining sections of this chapter describe such a front-end facility that acts as an interface between a user and a system. This facility is called the UI (User Interface) and consists of 1) programs that respond appropriately to a user's request for a user aid, 2) a set of primitive functions for communicating between the UI and a system, and 3) data structures required by the UI to provide user aids. These components are represented in Figure 5-5.

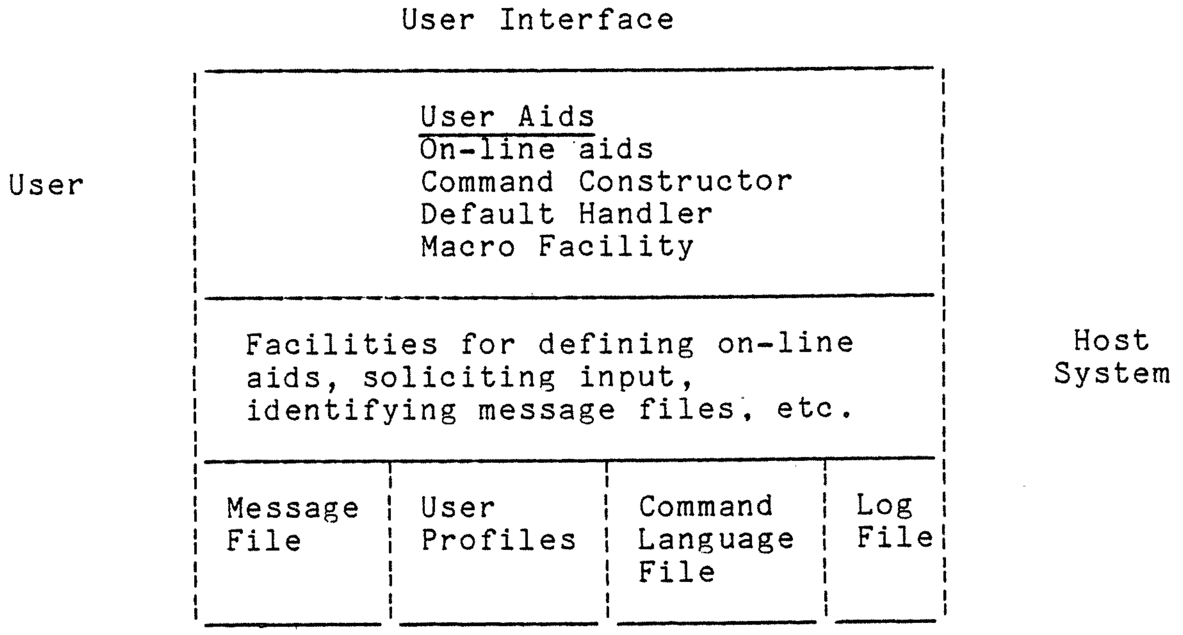

Figure 5-5. The User Interface

The <u>host system</u> is any application program (or the operating system itself) that communicates with a user through the UI. The <u>message file</u> contains all messages that can be displayed to a user to provide assistance or additional information. The structure of the message file is described below in section 5.2. The <u>user profiles</u> portion of the UI maintains information about users, their usage patterns, modes of communication, frequency of use, defaults (preambles), etc. The <u>command language file</u>

contains command language representations that are used to provide alternative modes of communication. The log file is used to record information on the usage of aids. Programs comprising the on-line aids portion of the UI have been implemented on a UNIVAC 1110 at the University of Wisconsin. The remainder of this chapter describes the FORTRAN functions used to communicate between a host system and the UI. With slight variations in calling sequences, argument transfer, and string handling, the same functions can be implemented in another language or on another machine. A similar interface, programmed in C, has been developed on a PDP 11/70 at the NASA-Ames Research Center.

5.2 Structure of the message file

Several on-line aids were proposed in section 4.2: EXPLAIN ERROR, EXPLAIN QUESTION, EXPLAIN TERM, HELP, MENU, and DOCUMENTATION. To provide these and similar aids, a system must establish an associated message file. The messages contained in this file are organized into scripts; the messages of a script correspond, for example, to the successively more detailed explanations of an error. Instead of (or in addition to) displaying a single message, a system may cause a script to be activated by the UI. An example of a script is:

```
File?
.
Please enter the name of the file you want to use.
If you don't have such a file, enter a valid file
name, and the system will create it for you.
.
A file name should be twelve characters or less.
Any of the following forms are valid:
   name.   name/readkey.   name/readkey/writekey.
```

The first message of this script consists of the single line 'File?'. A period in column one is used to mark the end of a message, and each successive message is displayed whenever the user invokes an appropriate user aid (in this case, perhaps EXPLAIN QUESTION).

On the UNIVAC 1110, any file may be partitioned into _elements_, each of which has a unique name. The 1110 operating system provides utility programs for handling such files; e.g., creating, finding, editing, or deleting any named element. Each script is stored and edited as such an element. A host system may activate any script by providing its element name to the UI. The UI uses the operating system's utility programs to access required scripts.

5.3 Primitive functions of the UI

The following primitive functions of the UI can be used to provide on-line aids:

UIOPEN: establishes communication between a host system

	and the UI
UIDEF:	defines a user aid, its invocation code, and the manner in which it is used
UICODE:	establishes more than one invocation code for an aid
UISET:	sets up a script to be associated with an on-line aid
UISHOW:	sets up a script and displays its first message
UINPUT:	solicits input from a user
UINDEX:	establishes an indexed script for *key-directed* aids (e.g., EXPLAIN TERM)
~~UILOG:~~	~~records on the log file data regarding use of~~ the aids
~~UILMSG:~~	~~records on the log file an arbitrary log message~~ created by the host system

5.3.1 Initiating communication: UIOPEN

When interaction with a host system is initiated, that host system must identify its message file to the UI by calling the primitive function

UIOPEN (*msgfile*, *errmsg*, *logfile*)

where *msgfile* is the name of the message file that will be accessed by the UI during subsequent interaction with the user.

There are many errors that may be made by a host system in calling the UI's primitive functions. We would like to preclude the display of a meaningless message to a user, e.g.:

```
/list 3,all,1967,9178

**INSUFFICIENT NUMBER OF SCRIPT PARAMETERS **
  REQUIRED IN 'NOREC'  CALLED FROM
  SEQUENCE NUMBER 07716 OF RANGETEST
```

For this reason, a system must provide a message to be displayed in case of a system error. The second parameter of the UIOPEN function, errmsg, identifies a script containing the desired "system error" message. The third parameter, logfile identifies a file where system errors are reported, for later inspection by the system's maintainers. For example, such a system error would appear to the user as follows:

```
/list 3,all,1967,9178

AN INTERNAL SYSTEM ERROR HAS BEEN ENCOUNTERED.  THIS
SHOULD NOT PREVENT YOU FROM CONTINUING.  PLEASE CHECK
YOUR COMMAND AND TRY AGAIN. OR CALL A CONSULTANT AT
262-6886.
```

The log file would contain the message

```
**INSUFFICIENT NUMBER OF SCRIPT PARAMETERS REQUIRED IN
  'NOREC' CALLED FROM SEQUENCE 07716 OF RANGETEST.
```

In addition to reporting internal system errors, the log file is used by the UI to record statistics on user aid invocations. This allows a system's maintainers to identify frequent user errors, frequently used aids, frequently referenced messages, and scripts with an insufficient number of messages. This information may result in changes to the system, its documentation, or its message file.

5.3.2 Defining aids: UIDEF and UICODE

Corresponding to each aid (EXPLAIN ERROR, EXPLAIN QUESTION, etc.) the following information must be provided to the UI by a host system:

1) a message to be displayed when no script has been established for an aid, but a user has invoked that aid;

2) a message to be displayed when a script has been exhausted, i.e., when successive invocations of an aid have caused all of a script's messages to be displayed;

3) an invocation code that, when entered by a user, causes the next message of a script to be displayed;

4) the type of aid being defined.

These four components of an aid are defined once; thereafter, any script may be associated with that aid. An aid is defined in the UI by calling the primitive function

UIDEF (*empty*, *exhstd*, *code*, *type*)

where *empty*, *exhstd*, *code*, and *type* correspond, respectively, to the four components described above. The value returned by this function is called the *aid identifier*, and is subsequently used to identify the aid with which a script is to be associated.

The _type_ parameter of UIDEF specifies how a script should be treated when something other than an aid's invocation code is entered. Three types of aids may be defined. If _type_ is 1, this indicates that the script is to be removed from the user aid; unless another script is established by a call to UISET, any request for that aid will result in the EMPTY message. This is used in aids like EXPLAIN ERROR and EXPLAIN QUESTION, where not requesting the aid indicates that the previous error message or question has been understood.

If _type_ is 2, the script is re-established when the user enters something that is not a request for a user aid. This is used in aids like MENU or HELP, wherein a long list of valid commands may be divided into several messages. Such scripts are generally established once, but remain available for the duration of a task.

If _type_ is 3, the user's position in the script remains unchanged when the user enters something other than a request for a user aid. Any subsequent request for that aid will cause the next message of the script (if one exists) to be displayed. For instance, this may be used for the EXAMPLE aid, to provide a new example each time one is requested. Another type of aid that may be provided with this feature is INSTRUCT. By establishing such an aid, and a corresponding script, a host system can provide instructions to a user. These instructions are displayable in conjunction with normal system use, and behave like the page-turner of an on-line introduction to using the system. A user may alternate between requesting instruction (by invoking the INSTRUCT aid) and entering valid commands as suggested by the given instructions. Similarly, a NEWS aid can be established to provide information about recent enhancements, discovered errors, and scheduled modifications in the host system.

For example, let us say that the following scripts exist in the message file:

script	contents
ERREMP	No error to explain.
ERREXH	No further explanation is available.
QEMP	No question to explain.
QEXH	No further explanation of this question is available.

The creation of aids for explaining errors and questions is effected by the following statements:

```
ERRAID = UIDEF ('ERREMP', 'ERREXH', '?ERR', 1 )
EXQAID = UIDEF ('QEMP', 'QEXH', '?EXQ', 1 )
```

The variables ERRAID and EXQAID are the <u>aid identifiers</u> that are subsequently used to refer to these aids; their existence in the UI is represented in Figure 5-6. Aliases for an aid are established by calling the function

UICODE(*aidid*, *alias*)

where *aidid* is the aid identifier of an existing aid and *alias* is another invocation code associated with that aid. This function is used, for example, so that an aid can be invoked by several different codes or function keys.

```
                  Invocation
ERRAID ------>    Codes          EMPTY:  No error to explain.
                 |__________|    EXHSTD: No further explanation
                 | ?ERR     |            is available.
                 | ?ERROR   |
                 |__________|    SCRIPT:

                  Invocation
EXQAID ------>    Codes          EMPTY:  No question to explain.
                 |__________|    EXHSTD: No further explanation
                 | ?EXQ     |            of this question is
                 | ?EQU     |            available.
                 |__________|
                                 SCRIPT:
```

Figure 5-6. Representation of two defined aids

5.3.3 Activating a script: UISET and UINPUT

A script is established for a user aid by the primitive function UISET, whose calling sequence is

UISET (*aid-id*, *scriptname*, *p1*, *p2*, ...)

where *aid-id* is an aid identifier (e.g., ERRAID or EXQAID in the example above) and *scriptname* is the name of the desired script. Before describing the other arguments of this function, we must describe the primitive function responsible for soliciting user input. The function

UINPUT (image)

is called to read in a user's input. If the input is one of the invocation codes established in the UI, the next message of that aid's current script is displayed. When the user enters something other than a request for a user aid, the input is transferred to the specified area, image, and the length of that input is returned as the value of the function. The value returned is negative if the user enters an end-of-file.

The parameters p1, p2, ... pn of the function UISET are arguments that may be used in conjunction with a script. By supplying such arguments, it is possible to vary the contents of a script's messages according to values known by the host system. For example, the system may wish to include as part of a message the user's name, the value that caused an error, the anticipated cost of a task, or a reference to relevant sections of documentation.

5.3.4 Script parameters, references, and special characters

In a script, parameters are denoted by an asterisk followed by a parameter number. For example, let us say that a script called NOREC consists of the following messages:

```
NO SUCH RECORD.

.
There is no record whose number is *1.
Valid record numbers in this database
must be between *2 and *3.
```

This script is activated by the UISET primitive as follows:

```
CALL UISET (ERRAID, 'NOREC', RNUM, MINREC, MAXREC)
```

where RNUM contains the incorrect value entered by the user; this is substituted for *1 in the script if the user invokes the user aid identified by ERRAID. The values of MINREC and MAXREC are similarly substituted for *2 and *3 in the same script.

Messages may refer to other scripts in the message file in one of two ways. A plus sign (+) in column one, followed by a script name, indicates that the currently active script should be suspended. Subsequent messages will be constructed from the named script, and when that script is exhausted, the current script will be reactivated. This is comparable to a subroutine facility among scripts. The UI behaves exactly as though the named script appeared in place of the referencing plus sign and script name. A "greater-than sign" (>) in column one, followed by a script name, indicates that the named script should be activated in place of the currently active script.

```
Script          Contents
------          --------
NOTTHERE        String not found.
                .
                "*1" does not appear in this line,
                which begins with "*2 ..."
                .
                >CHANGEDOC

MISSDELIM       Missing Delimiter.
                .
                The string you wish to change should be
                separated from its replacement by the
                change delimiter, for which you have used *1
                .
                >CHANGEDOC

CHANGEDOC       For a detailed description of the change
                command, see section 2.3 of the EDIT
                reference manual.
                .
                +MANUAL

MANUAL          The EDIT manual is available from the
                documentation center for $3.45

DELETELINE      Delete? (Y or N)
                .
                +YES
                line *1 to be deleted.
                +NO
                .
                >DELETEDOC

YES             Enter "Y" if you wish &

NO              Otherwise, enter "N"

DELETEDOC       For a detailed description of the delete
                command, see section 2.5 of the EDIT
                reference manual.
                .
                +MANUAL
```

Figure 5-7. Sample scripts

Three other characters have special meaning in a script. An ampersand (&) at the end of a line causes the following line (which may be in another script) to be concatenated with the current line before being displayed. A hash mark (#) in column one causes a line to be ignored; this allows comments to be included in scripts for purposes of maintaining a message file. A single quote (') preceding any special character (., *, +, >, &, or #) indicates that the character should be treated as a displayable character, rather than serving as a message delimiter, parameter marker, script reference, line concatenator, or comment delimiter, respectively. Figure 5-7 shows several scripts from a message file that use the capabilities just described. Figure 5-8 represents a hypothetical session in which the UI provides assistance as prescribed by these scripts.

5.3.5 Initial display of a message: UISHOW

When a script is established by calling UISET, no message is displayed. To establish a script <u>and</u> display its first message, the host system must call the primitive function

UISHOW (<u>aid-id</u>, <u>scriptname</u>, <u>p1</u>, <u>p2</u>, ... <u>pn</u>)

where <u>aid-id</u>, <u>scriptname</u>, and <u>p1</u> through <u>pn</u> are the same as for the primitive function UISET.

```
change *theire*there
String not found.

?ERR
"theire" does not appear in this line,
which begins with "if their are none ..."

?ERR
For a detailed description of the change
command, see section 2.3 of the EDIT
reference manual.

change *their*there
ok.

delete 157
Delete? (Y or N)

?EQU
Enter "Y" if you wish line 157 to be deleted.
Otherwise, enter "N"

?EQU
For a detailed description of the delete
command, see section 2.5 of the EDIT
reference manual.
```

Figure 5-8. Sample session using scripts in Figure 5-7

5.3.6 Key-directed aids: UINDEX

Several types of on-line aids require that a user specify a term or number together with the aid's invocation. The DOCUMENTATION aid, for example, requires specification of a section number. In like fashion, EXPLAIN TERM requires the term for which the user wants a definition or explanation. We shall call such aids _key-directed_ aids, indicating that the information obtained is determined by a _key_ provided by the user, e.g., a word, phrase, or number.

Key-directed aids are provided in the UI through the message file, but with a different script structure and primitive function for defining the aid.

A key-directed aid requires an index that associates scripts with user-specifiable keys. This index is a script in the message file, whose format is

```
key1
key2
key3
+scriptname1
.
.
.

keym
+scriptnamen
```

Each index entry consists of a set of keys followed by the name of a script to be associated with any one of those keys. A one-to-one correspondence between keys and scripts would be too restrictive. An index allows the host system's implementer to provide for synonyms, variant forms, or phrases that would not be valid script names.

Figure 5-9 includes a portion of the index script for the DOCUMENTATION aids. Also shown are some of the scripts referenced by that index. The documentation is taken, in part, from the reference manual of a text-editing system at the University of Wisconsin. As with any other script, an index script may include comments (designated by a period in

```
Script    Contents

DOCX      9
          9.
          CHAPTER 9
          CHAPTER NINE
          HEADINGS
          FOOTINGS
          +CHAP9
          .
          9.1
          9-1
          HEADING SPECIFICATIONS
          +SEC9-1
          .
          9.2
          9-2
          TERMINATOR
          HEADING TERMINATOR
          +SEC9-2
          .
          TABLE OF CONTENTS
          CONTENTS
          +CONTENTS
          .
          INDEX
          +INDEX


CHAP9     Chapter 9: Headings and Footings
          9.1 Heading Specifications
          9.2 Terminator
          9.3 Heading Text
          9.4 Heading Placement in a Text
          9.5 Page Numbering


SEC9-2    9.2  Terminator
          A heading specification is terminated by an átend
          format command.  This command marks the end of the
          heading begun by the preceding heading command; it
          may be followed by another heading command or by
          lines of the main text.
```

Figure 5-9. Sample scripts for a key-directed aid

Script	Contents
CONTENTS	1. INTRODUCTION 2. Using the MACC Computer 3. Getting Started in TEXT 4. TEXT Format Commands 5. TEXT Editing Commands . . .
INDEX	abbreviate add alert backspacing backscore break change clear codes copy crashes continuous forms
BADDOC	There is no documentation available for the section or term you specified.
NODOC	You must provide a section number or term immediately after ?DOC. For more information, enter ?DOC again. . To obtain a list of valid chapter numbers, you may enter ?DOC CONTENTS. For a list of valid terms, you may enter ?DOC INDEX.
BADTRM	No explanation available for this term or command.
NOTERM	You must provide a term or command. For a list of valid terms, enter ?TERM VOCAB.

Figure 5-9 (continued)

column one). In the index script of Figure 5-9, a period is used to separate each group of keys.

A key-directed aid is established in the UI by calling the primitive function

UINDEX(*aidid*, *indexscript*, *noinfo*, *nokey*)

Aidid identifies the aid that, together with a key, causes the UI to provide the requested information. *Noinfo* is the name of a script that contains a message to be displayed to the user in case the key entered has no information associated with it. The argument *nokey* identifies a script (with possibly several messages) that should be used when a user enters an invocation code without a key. This script may suggest, for example, valid terms or the correct way of using the aid. For example, the DOCUMENTATION aid, in conjunction with the scripts in Figure 5-9, is established as follows:

```
CALL UINDEX( DOCAID, 'DOCINDEX', 'BADDOC', 'NODOC')
```

5.3.7 Monitoring system usage: UILOG and UILMSG

The function UILOG writes a usage summary on the log file associated with a host system. The summary includes the number of times each aid was established and requested, how many times it was empty or exhausted, and what terms were requested for key-directed aids. These data can be used to identify aids or scripts that should be improved, as well as host system features that are unclear to users. The form of the function is

UILOG(*logid*)

where _logid_ is a character string associated with the recorded date. The _logid_ can be used, for example, to identify different program modules or classes of users. The function UILMSG can be called by the host system to record an arbitrary message on the log file.

5.4 Summary

This chapter has described the implementation of a user interface that is adaptable to a wide range of operating systems. The primitives comprising the user interface enable any interactive system to provide aids that are unobtrusive and situation-dependent. The choice of an appropriate aid and the depth of corresponding explanations may be left to the user. In addition to encouraging greater consistency among the subsystems that make up a larger system, the primitives make it possible to 1) easily vary the protocols and content of on-line aids, and 2) monitor the use of on-line aids.

6. Evaluating the Effectiveness of User Aids

The front-end facilities described in the previous chapter provide the uniformity lacking in other systems' aids. The on-line aids portion of the front-end was readily incorporated into two interactive applications at the University of Wisconsin: a database management system pertaining to state elections, and a software distribution system used by the computing center. However, several questions remained unanswered regarding the effectiveness and benefits of on-line aids, the cost of providing on-line aids in a system, and the manner in which aids should be provided for a given application. This chapter describes an experiment that was conducted with a third system to investigate the relationship between on-line aids and ease of use, as measured by user performance and satisfaction. Development of the experimental system also provided a setting for studying the effort and computing resources required to incorporate on-line aids into a program.

The empirical study of user behavior in an on-line computing environment has received relatively little attention. In one of the earliest studies of programmer performance, Grant and Sackman found that time-sharing was more effective than batch facilities for debugging purposes [GR67]. In their study, as well as others [AD69, SC67, SM67], time-sharing was found to be generally superior with respect to programming time and programmer preferences, while batch use was less costly in terms of computing resources. In an often cited study by Miller, the reasonableness of system response times was found to depend on what users perceived as task closure [MI68].

While these studies were primarily concerned with programming activities, controlled experiments have also been conducted in more general interactive environments. Walther and O'Neill studied the effects of user, terminal, and language characteristics on user performance with a text editor [WA74]. They found that performance was significantly affected by the interaction of these factors, rather than any single factor. Hansen compared the performance of non-programmers solving complex problems in on-line vs. batch environments, and concluded that on-line use exhibited some advantages [HA76]. Miller investigated the effects of output display characteristics on user performance and user satisfaction [MI77]. Changes in display rates (1200 baud vs. 2400 baud) did not have significant effects on user performance and attitudes of users, while significant effects did result from changes in output display variability.

The experiment described in this chapter, like the studies above, was concerned with user performance in an on-line environment. Although no particular emphasis was placed on programming activities per se, the questions addressed by this study apply to any interactive task, including programming.

6.1 Design of the experiment

The purpose of the experiment was not to define an all-encompassing model of man-machine interaction, but to focus on one important aspect thereof: the effects of differing supportive capabilities on ease of use. The ease with which a system can be learned and used depends on many factors, e.g., how frequently it is used, experience levels of its users, the language and protocols used to communicate with the system, including terminal characteristics, and the methods used for training and support. To study all of these factors and their interaction in a single experiment would be prohibitively complex and unwieldy. The approach taken, therefore, was to design an experimental setting in which a single independent variable would be varied while all other factors would remain as fixed as possible. A one-way analysis of variance was used to determine whether any differences in the dependent variables representing ease of use could be attributed to the different treatments. A detailed discussion of the statistical methods used in the experiment can be found in the literature [CO57, HA73b, KI68].

The system used in the experiment had to be simple enough to be learned and used by subjects in a short period of time. In addition, it had to be sufficiently representative of other systems that any conclusions of the study could be extended to a broader category of interactive systems. Finally, its implementation had to provide the ability to vary the properties of the independent variable under study. Because no available system met these requirements, a simple bank account management system, called TELLER, was implemented specifically for the experiment. A 37-page user manual was written for this system (see Appendix A for an overview). The system was designed to represent existing transaction-oriented systems: reservation systems, text-processing systems, special-purpose database management systems, home computing systems, and the like. In addition, its command language and task hierarchy were conceptually similar to most operating systems' job control languages. This similarity was deliberately increased by adding a few anomalies, inconsistencies, and syntactic restrictions.

6.1.1 Hypotheses

The general hypothesis of the experiment was that ease of use is affected by the provision of on-line aids and the manner in which those aids are provided. As a result of the research described in previous chapters, it was deemed important that on-line aids should be

1) unobtrusive -- making it possible for users to obtain assistance with minimal digression from the task at hand,

2) multi-level -- allowing users to obtain additional or successively more detailed information,

3) situation-dependent -- providing information appropriate to the context in which the user seeks assistance, and

4) function-specific -- enabling a user to select exactly that type of assistance required (e.g., format descriptions, examples, or explanations of an error).

It is common to find systems whose facilities for on-line assistance receive very little use [MA75b, BO74a]. One reason for this phenomenon is, perhaps, that such facilities are inadequate or cumbersome to use. The first hypothesis of the experiment was:

> On-line aids that are unobtrusive, multi-level, situation-dependent, and function-specific are more likely to be used than on-line aids that do not have these properties.

Henceforth, the term on-line aids will be used to refer to aids that have the four properties described above. The second hypothesis of the experiment was:

A system that includes on-line aids is easier to use than one that does not.

6.1.2 The independent variable

The independent variable chosen for the experiment was assistance during use of a system, in the form of tutorial information, command descriptions, explanations of messages, and examples. These categories of assistance can be easily provided by the on-line aids portion of the user interface, described in chapter 5. It should be noted that modes of communication, default structures, and language extension facilities were <u>not</u> varied or formally investigated in the experiment. However, these and other factors were fixed at levels representing a broad class of systems. Four treatment groups were planned for the experiment:

1) the NA (no aids) group; subjects who would have no access to (or knowledge of) any on-line aids for the system;

2) the H (HELP) group; subjects who would have access to a HELP mode, similar in design to the HELP facilities of most existing systems;

3) the OA (on-line aids) group; subjects who would have access to a set of on-line aids that were unobtrusive, multi-level, situation-dependent, and function-specific

4) the NM (no manual) group; subjects who would have access to the same on-line aids as the OA group, but would be required to use the system without a manual.

Except for these different treatments, all subjects first learned about the system from the manual, used the same system, and performed the same set of tasks. Only the NA, H, and OA groups were allowed to use the manual while interacting with the system. The HELP mode and on-line aids designed for the H and OA groups, respectively, are described in Appendix B.

6.1.3 The dependent variables

Several dependent variables representing ease of use were investigated. The desire to make systems easy to use is largely motivated by an economic concern: making the most effective use of human and computer resources. Cost-related measures of user performance were therefore included as dependent variables. Objective measures alone do not necessarily reflect long-term effects (e.g., how likely it is that users will continue to use a system) or effects on human factors (e.g., stress, confidence and performance degradation). Several researchers have reflected these concerns in calling for more 'humanized' or 'user-friendly' systems [DZ78, GI77, KE74, KL73, LI60, ST74]. To represent these aspects of ease of use, therefore, a subjective measure of user satisfaction was also included as a dependent variable.

Objective measures of ease of use were obtained by monitoring user performance. The following items were recorded: computing costs incurred in completing all tasks, the number of lines entered by a user, the number of lines displayed by the system, the total time required to complete all tasks, and the number of errors made. These items reflected the effectiveness of subjects in using computing resources and human resources (principally, time). The billing system used at the University of Wisconsin applies separate charges for central processor time, mass storage I/O, executive requests, and other computing resources. It was felt that total computing costs incurred would provide a cumulative measure of computer resource utilization. Errors were divided into two categories: unique errors and repeated errors. An error was recorded as unique the first time it was made in a session. Each time the same error was made in the session it was recorded as a repeated error. For example, if a subject made three errors in performing some single task, this was recorded as one unique error with two repetitions.

A subjective measure of ease of use was provided in the form of a questionnaire filled out by subjects after they used the system. The questionnaire, shown in Appendix C, included ten questions pertaining to characteristics of the system and its use; e.g., how easy it was to enter commands, the clarity of system messages, and how easily the system could be used without human assistance. Each question had five possible responses, ranging between 1 (for very hard to learn, use, or understand) and 5 (for very clear or easy to use). The questions were given equal weight, and the sum of all ten responses was taken to represent a subject's overall evaluation of ease of use.

6.2 The pilot study

The experiment was preceded by a pilot study, whose purpose was to identify and correct problems that might arise in the experiment or its attending analyses.

6.2.1 Objectives

The pilot study had four main objectives: 1) to evaluate the suitability of the manual, system, and prescribed tasks to the experiment, 2) to determine how many subjects would be required in the experiment, 3) to identify areas bearing closer study in the experiment, and 4) to gain familiarity with the mechanics of conducting the experiment.

The experiment was intended to last about 75 minutes per subject. Within that time, each subject was expected to learn about the system from the manual, gain some experience with the system, and deal with any errors that might be encountered. The pilot study addressed several related questions: Were there any errors in the manual, the system, or the on-line aids? Were there any inconsistencies among the manual, the system, the on-line aids, or the written description of tasks to be performed? Was it reasonable to expect subjects to complete the designated tasks in the time allotted? The answers to these questions would determine whether any modifications would be required for the actual experiment.

The goal of the experiment was to study the effects of four different treatments on perceived ease of use and user performance. When differences between experimental treatments exist, their detection requires an adequate sample size. If too small a sample is used, any effects of the different treatments may be obscured by differences between individuals. Standard statistical procedures can be used to determine required sample size, given a population error variance and desired degree of confidence. A goal of the pilot study, therefore, was to provide estimates of the treatment means and error variance for each of the dependent variables to be studied in the experiment.

6.2.2 Subjects

Subjects were drawn from an introductory computer science course at the University of Wisconsin. This course is intended for non-computer science majors, and provides a one-semester introduction to how computers work, communicating with computers, areas of application, and simple BASIC programming. Students in this course have typically not had any previous programming experience. It was felt that subjects from this course would be representative of so-called naive users, and that this homogeneity would minimize variations due to prior experience. To limit further extraneous variation, all subjects were processed during the second week of class (after three to six lectures). Subjects were randomly assigned to the four treatments: the NA (no aids), H (HELP mode), OA (on-line aids), and NM (no manual) groups. Each group had six subjects.

Although participation in the experiment was not a course requirement, subjects were told that they would gain some familiarity with interactive use of a computer and an exposure to some of the concepts and problems related to such use. One purpose of the pilot study was to see whether this would be sufficient motivation for completing the experiment in a reasonable fashion.

6.2.3 Procedure

Each subject was tested individually. The instructions (shown in Appendix D) were read to a subject, who was then given a manual describing the TELLER system. Subjects were asked to spend about 30 minutes reading the manual to learn about the system. When a subject had finished reading the manual, he was seated at a terminal and asked to perform the tasks shown in Appendix E. All subjects used the same terminal, a DECWRITER operating at thirty characters per second. When a subject had completed the designated tasks or had taken so much time that he no longer wanted to continue, he was asked to fill out a questionnaire (shown in Appendix C).

6.2.4 Results and interpretation

The results of the pilot study are summarized in Appendix F. The fourth group was excluded from the experiment when it became obvious that subjects in this group could not use the system without a manual. These subjects spent their entire time on the first few tasks and required continual help from the experimenter. With little variation, all other subjects read the manual in about thirty minutes, and required human assistance two or three times while performing tasks. The fourth group _did_ spend the same initial thirty minutes reading the manual, and knew that they would be asked to perform the tasks without the manual.

Perceived ease of use was measured by the sum of responses to the questionnaire (see Appendix F, column TQS). A one-way analysis of variance on total questionnaire scores (see Table 6-1) indicated that the differences between groups were not statistically significant. That is, differences in group means could not be attributed to treatment effects.

Source	df	SS	MS	F-ratio
Treatments (between groups)	2	76.8	38.4	1.53
Error (within groups)	15	375.5	25.0	
Totals	17	452.3		

Group	number of subjects	Mean	St. dev.
NA (no aids)	6	42.33	1.97
H (HELP mode)	6	39.17	6.55
OA (on-line aids)	6	37.33	5.32

~~Pooled st. dev.: 5.0~~

Table 6-1. Analysis of variance for perceived ease of use

Comparisons based on performance-related statistics could not be made, because no one in the third group (OA), which had on-line aids available, completed all the tasks. All of the subjects in group 1 (no on-line aids available) and all but one in group 2 (HELP mode available) completed the tasks in times ranging between 36 and 68 minutes. Because the tasks completed were different among subjects, it was not possible to normalize performance data on a per-task basis.

It had already been observed in the course of the experiment that subjects in the OA group preferred to use the manual rather than the aids. Even when human assistance was requested and the experimenter suggested using the aids, subjects were reluctant to do so. At least three subjects said that they viewed the aids as "a last resort." Furthermore, subjects in the OA group appeared to be having more trouble using the system than those in other groups.

To understand why subjects in the OA group had so much difficulty, the protocols from their sessions were examined. In the few cases where an aid was used, it was used incorrectly or the wrong aid was selected. However, a more serious cause of errors was the haphazard use of the system in a trial-and-error mode. A command would be entered; if an error resulted, a slightly different syntax or a different command was entered; after several repeated errors, the manual was consulted. In some cases, commands were entered that had no resemblance to anything described in the manual. Thus, it was not surprising that no one in the OA group completed all the tasks.

Drawing conclusions from only six subjects is questionable, but it seems that the following conditions in combination contributed to the exceptionally poor performance of subjects in the OA group:

1) subjects relied heavily on the manual for assistance,

2) subjects were unable to use the aids effectively,

3) the existence of the aids distracted subjects from the system's basic capabilities, or made it more difficult to learn those capabilities, and

4) the existence of the aids gave subjects the impression that they could enter almost anything and the system would guide them through successful completion of a task.

If users were so dependent on the manual, but given to believe that the system would help them out of any problem, what differences might result in their evaluation of the system with respect to the manual? Question 8 of the questionnaire provided some measure of this possible relationship:

Can you use the system without referring to the manual?

1. I can't use the system without constantly referring to the manual.
2. I would need to refer to the manual frequently.
3. I would need to refer to the manual occasionally.
4. I can use the system with very little reference to the manual.
5. I can use the system without the manual.

This question also stood out because in all three groups its mean response value was lower than any other question's. An analysis of variance, summarized in Table 6-2, showed a significant treatment effect ($p<.05$) for responses to this question. The existence of on-line aids (or their effect on user performance) caused subjects in the OA group to have less confidence in their ability to use the system without a manual.

Source	df	SS	MS	F-ratio
Treatments (between groups)	2	5.444	2.722	4.54 *
Error (within groups)	15	9.000	.600	
Totals	17	14.444		

Group	number of subjects	Mean	St. dev.
NA (no aids)	6	3.167	.753
H (HELP mode)	6	2.667	.516
OA (on-line aids)	6	1.833	.983

Pooled st. dev.: .775

* $p<.05$

Table 6-2. Analysis of variance for response to question 8

6.2.5 Summary

All the goals of the pilot study were met except one: determining by statistical procedure how many subjects would be required in the experiment. The manual was found to be clear and consistent with the system, the instructions, and the on-line aids. However, subjects from the introductory computer science course could not be expected to learn about the TELLER system and then use it to perform the designated tasks, all within about seventy-five minutes. Furthermore, subjects from this population relied heavily on the manual, and even when provided with on-line aids could not use the system without the manual.

No significant difference was observed in subjects' perceived ease of use between the three experimental treatments. The provision of on-line aids did have an adverse effect on user performance, and subjects who used the aids had significantly less confidence in their ability to use the system without a manual. It appeared that at the very least, subjects in the OA group would require a demonstration of the on-line aids, in order to become familiar with their correct use.

6.3 The experiment

6.3.1 Hypotheses and subjects

The objectives of the experiment remained the same: to determine whether any differences in perceived ease of use or user performance would result from the provision of on-line aids. The most significant change resulting from the pilot study was in the choice of subjects. The pilot study demonstrated that completely inexperienced subjects were too dependent on a manual while using the system, were confused by the provision of on-line aids and could not complete the prescribed tasks in a reasonable time. In addition, it was felt that their lack of exposure to other systems contributed to their invariant view that the system was easy to use. Subjects for the experiment were therefore drawn from a different population: computer science students who had had some previous experience with interactive systems on the UNIVAC 1110 at the University of Wisconsin.

Because the pilot study had revealed significant differences between treatments in response to question 8, and because those differences were thought to be a result of subjects' inexperience, a third hypothesis was addressed by the experiment:

> For users with prior computing experience, the provision of on-line aids increases confidence in the ability to use a system without a manual.

To provide adequate motivation and ensure that all subjects would complete the experiment in a reasonable fashion, subjects were paid five dollars to participate. Volunteers were drawn from intermediate and advanced computer science courses, with the requirement that they had used the UNIVAC 1110 interactively. Thirty subjects participated in the experiment; all but two were computer science majors; fourteen were graduate students and sixteen were undergraduates. Subjects were randomly assigned to the three treatments remaining after the pilot study: the NA (no aids), H (HELP mode), and OA (on-line aids) groups. All subjects were allowed to use the manual; the NM (no manual) group originally intended for the experiment was not tested.

6.3.2 Procedure

Each subject was tested individually and the system used was the same as that in the pilot study. The questionnaire also remained unchanged. The following procedural changes were made as a result of the pilot study's findings:

1) Subjects were asked to spend about twenty minutes reading the manual to learn about the system, rather than the thirty minutes allotted in the pilot study.
2) There were slightly fewer tasks than in the pilot study, and instructions for using the terminal were removed (see Appendix G).
3) In the case of the H and OA groups, the HELP mode or on-line aids, respectively, were demonstrated before a subject was allowed to begin performing the tasks.

Because of these differences in procedure, a rigid statistical comparison between results of the experiment and those of the pilot study is not possible. At best, general differences between the two experimental settings may be identified. In the following discussion, any conjectures regarding such differences are intended solely as questions deserving further study in a more controlled setting. Indeed, the overall difference in results between the pilot study and the actual experiment suggests that previous experience is an important independent variable whose interaction with on-line aids bears closer investigation.

6.3.3 Results and interpretation

The results of the experiment are summarized in Appendix H. Four subjects in the OA group did not use the manual at all. For this reason, question 6 ("How effective is the manual while using the system?") had to be discarded. Perceived ease of use was measured by the sum of responses to questions 1 through 5 and 7 through 10. Only three subjects in the H group used the HELP mode, so no formal statistical comparison was made between the HELP mode's ease of use and that of the on-line aids (as represented by questions 11 and 12). All subjects completed the tasks; even those who took longer than the intended 75 minutes finished.

The remainder of this section analyzes the effects of the three treatments on

1) user confidence, as represented by responses to question 8,

2) overall perceived ease of use, as represented by the sum of responses to the first ten questions,

3) effectiveness and ease of using the aids, as represented by responses to questions 11 and 12, and

4) user performance, as represented by
   a) the total cost of performing tasks,
   b) total connect time, and
   c) the number of unique and repeated errors.

Source	df	SS	MS	F-ratio
Treatments (between groups)	2	12.6	6.3	10.25 *
Error (within groups)	27	16.6	.615	
Totals	29	29.2		

Group	number of subjects	Mean	St. dev.
NA (no aids)	10	3.0	.471
H (HELP mode)	10	3.3	1.059
OA (on-line aids)	10	4.5	.707

Pooled st. dev.: .784

* p<.01

Table 6-3. Analysis of variance for response to question 8

The relationship between on-line aids and confidence in an ability to use the system without a manual (as represented by question 8) was exactly opposite of that observed in the pilot study. An analysis of variance on mean scores for question 8 (see Table 6-3) showed a significant treatment effect ($p<.01$). This time, however, those who used the on-line aids were more confident than those who did not use them. A post-hoc comparison of means, using Tukey's HSD test, showed that significant differences existed between the NA and OA groups ($p<.01$) and between the H and OA groups ($p<.01$).

It is worth noting that for the NA and H groups, the mean of responses to question 8 was lower than that of any other question, as was the case in the pilot study for all three groups. We can only offer the conjecture that beginning users of a system are generally not confident that they can use the system without a manual. The effectiveness of on-line aids in gaining that confidence seems to depend on the nature of the aids <u>and</u> how much previous experience a user has had with interactive systems.

Perceived ease of use, as measured by the sum of responses to the first ten questions (except question 6), did not differ significantly across the three groups. The analysis of variance on total questionnaire scores is shown in Table 6-4. The fact that on-line aids were not observed to affect perceived ease of use may reflect the inadequacy of the questionnaire as a measurement tool. In addition, user perception of a system's ease of use generally evolves over a much longer period of usage than that of the experiments. Finally, if on-line aids actually have an effect on perceived ease of use (and this experiment failed

Source	df	SS	MS	F-ratio
Treatments (between groups)	2	39.5	19.7	1.90
Error (within groups)	27	279.9	10.4	
Totals	29	319.4		

Group	number of subjects	Mean	St. dev.
NA (no aids)	10	38.10	2.64
H (HELP mode)	10	36.90	3.84
OA (on-line aids)	10	39.70	3.06

Pooled st. dev.: 3.22

Table 6-4. Analysis of variance for perceived ease of use

to detect such an effect), we must consider how strong that effect is. If the effect is small with respect to other factors' effects, detecting the effect requires a larger sample size. The TELLER system in particular had several other features that undoubtedly contributed to variations in perceived ease of use, e.g., command abbreviations, prompts, alternative command formats, and abbreviated 'menus' of valid responses. For such a subjective attribute as ease of use, more precise instruments and quantitative measures must be developed. The work of Dzida et al [DZ78] is a promising step in this direction.

	Q11	Q12
Group H (HELP mode)	*	*
	*	*
	4	4
	*	*
	*	*
	*	*
	*	*
	*	*
	4	5
	4	4
Group OA (on-line aids)	4	5
	5	5
	5	5
	5	5
	5	5
	5	5
	5	5
	5	5
	5	5
	5	5

* indicates that HELP mode was not used at all

Table 6-5. Perceived effectiveness of aids

Responses to questions 11 and 12, regarding the perceived effectiveness of the HELP mode and on-line aids, give every indication that the on-line aids were more useful (see Table 6-5). Only three subjects in the H group used the HELP mode; every subject in the OA group used the aids, albeit to varying degrees and in different ways. No subject in the H group tried to use the system without the manual; four of the ten OA subjects were able to use the system without the manual. Although the responses to questions 11 and 12 were generally high (4 or 5) in both groups, all three subjects who used the HELP mode judged it to be "usually helpful" (4), while all but one of the OA subjects felt that the on-line aids were "very helpful" (5). Similarly, two of the three H subjects indicated that the HELP mode was "usually easy to use", while all of the OA subjects felt that the aids were "very easy to use."

These data give some support to one of the hypotheses of the experiment: that the degree to which on-line aids are used depends on the manner in which they are provided. Specifically, the content of the messages was generally the same in the HELP mode and the on-line aids; the fact that the on-line aids were unobtrusive, multi-level, situation-dependent, and function-specific appears to have increased their usage and their perceived effectiveness.

The differences in total computing costs between groups were found to be small and not statistically significant. The analysis of variance for total costs is summarized in Table 6-6. These cost differences reflect, in part, peculiarities of the TELLER system's implementation and the rate structure of the University of Wisconsin's computing center. To the extent that the computing center's rates and the system's implementation represent a broader category of systems, it can be concluded that using on-line aids does not significantly affect computing costs. Cost-related factors of the TELLER system's implementation are summarized below in section 6.4.

Source	df	SS	MS	F-ratio
Treatments (between groups)	2	.2205	.1102	1.85
Error (within groups)	27	1.6102	.0596	
Totals	29	1.8306		

Group	number of subjects	Mean	St. dev.
NA (no aids)	10	1.500	.149
H (HELP mode)	10	1.688	.220
OA (on-line aids)	10	1.513	.329

Pooled st. dev.: .244

Table 6-6. Analysis of variance for total cost of performing tasks

The most dramatic difference in user performance was in the time required to perform the designated tasks. The mean connect time for the NA group was about 42 minutes, that for the H group was about 38 minutes, and that for the OA group was about 30 minutes. An analysis of variance on mean connect time (see Table 6-7) indicates that these differences were due to treatment effects ($p<.05$). A post-hoc comparison of means using Tukey's HSD test showed that differences in connect time were significant between the NA and OA groups. The provision and use of on-line aids enabled subjects to complete all the tasks in significantly less time. What is not known from the data is how much of this time difference was due to learning and how much was a result of speedier access to material through on-line documentation as opposed to the written manual, its index, and appendices. Furthermore, faster performance by itself is an inadequate measure of overall user performance. The time to complete a task or set of tasks must be weighed against the number of errors that are made, their persistence, and their seriousness.

In an experiment where errors are treated as a dependent variable, it is important to consider how errors should be categorized and which categories should be included in the analysis. Studies of programmer behavior or program correctness have generally classified errors as syntactic or semantic, minor or major logical errors, and coding-related or debugging-related errors [BO74b, GO74, LO77b, RE77, YO74]. For this experiment, only obvious keying errors and data errors (e.g., entering the wrong date) were excluded from the analysis. Separate tallies were kept for unique and repeated errors; for example, if a command was incorrectly entered three times, it was considered as one unique error with two repetitions.

Source	df	SS	MS	F-ratio
Treatments (between groups)	2	902.1	451.0	4.85 *
Error (within groups)	27	2509.1	92.9	
Totals	29	3411.2		

Group	number of subjects	Mean	St. dev.
NA (no aids)	10	42.28	11.67
H (HELP mode)	10	38.02	10.36
OA (on-line aids)	10	29.12	5.94

Pooled st. dev.: 9.64

* p<.05

Table 6-7. Analysis of variance for mean connect time (in minutes)

The analyses of variance for total errors, unique errors, and repeated errors are summarized in Table 6-8. No significant differences were observed for total number of errors or unique errors, but a significant treatment effect was found for repeated errors (p<.05). A post-hoc comparison indicated a significant difference in repeated errors between the H and OA groups (p<.05). This finding provides some evidence that the use of on-line aids is more effective than a manual alone in reducing repeated errors.

(Total number of errors)

Source	df	SS	MS	F-ratio
Treatments	2	90.6	45.3	2.19
Error	27	558.9	20.7	
Total	29	649.5		

Group	subjects	Mean	St. dev.
NA (no aids)	10	5.60	2.27
H (HELP mode)	10	9.80	6.76
OA (on-line aids)	10	7.10	3.35

(Unique errors)

Source	df	SS	MS	F-ratio
Treatments	2	46.87	23.43	2.85
Error	27	221.80	8.21	
Total	29	268.67		

Group	subjects	Mean	St. dev.
NA (no aids)	10	3.90	1.10
H (HELP mode)	10	6.60	3.78
OA (on-line aids)	10	6.50	3.03

(Repeated errors)

Source	df	SS	MS	F-ratio
Treatments	2	34.07	17.03	3.89 ($p<.05$)
Error	27	118.10	4.37	
Total	29	152.17		

Group	subjects	Mean	St. dev.
NA (no aids)	10	1.70	1.49
H (HELP mode)	10	3.20	3.19
OA (on-line aids)	10	.60	.84

Table 6-8. Analyses of variance for errors

Error rates -- e.g., total number of errors per minute and unique errors per minute -- could have been derived from the data and analyzed in the same manner as the unnormalized errors. However, these measures were seen as inadequate reflections of learning. More useful (but unfortunately not available from this experiment) would be a time-series analysis, reflecting how errors or error rates change over the course of sustained system usage. In addition, the experimental nature of the TELLER system precluded any weighting of errors with respect to their seriousness. While the experiment provides some indication that on-line aids affect error-related measures, it is clear that a more comprehensive taxonomy of errors must be developed.

In applying Tukey's HSD test to the dependent variables that exhibited treatment effects, no significant difference was found between the NA and H groups. In general, providing the HELP mode had no effect on user performance. Given the fact that seven of the H subjects did not use the HELP mode, they could be considered NA subjects. Accordingly, these seven subjects were pooled with the NA subjects, and the dependent variables representing ease of use were compared between this group (called the NA' group) and the OA group. The results, summarized in Appendix I, produced only one change: a significant difference due to treatment effects was indicated in the number of unique errors ($p<.05$). That is, although OA subjects generally repeated errors less often, they made more unique errors. This finding suggests that when on-line aids are provided and used, new users are more likely to make an error, but less likely to repeat it. We would caution again that the really important implications of this finding depend on data not investigated in this experiment: long-term persistence of errors, their seriousness, and their effect on learning.

6.4 Incorporating on-line aids: the programmer's view

A peripheral objective of the experiment was to study the effectiveness of the on-line aids facility (henceforth called AIDS) in terms of programming flexibility and resource requirements. While the AIDS facility was particularly well-suited to implementing the types of aids used in the experiment, ad hoc techniques could have been used to provide similar aids. Use of the AIDS facility was therefore observed with respect to

1) resource requirements: how much primary storage (for supporting programs) and mass storage (for messages) were required to provide on-line aids,
2) programmer's ease of use: how much effort was required to incorporate on-line aids into a program, and
3) facility limitations: what capabilities were required, but not provided by the AIDS facility.

Although the following observations pertain to the implementation of the TELLER system, it is felt that they apply to other systems of similar size and complexity.

The TELLER system was implemented in FORTRAN on a UNIVAC 1110. Database capabilities were incorporated through a FORTRAN interface to MISER, a general-purpose hierarchical database management system available at the University of Wisconsin. The entire system required 62,009 words of storage, broken down as follows:

TELLER main programs:	18,288 words
FORTRAN-MISER interface:	37,982 words
AIDS facility:	5,739 words

The messages for the on-line aids consisted of 171 scripts totalling 1813 lines; approximately 57,000 characters of disk storage were required for these messages and attending directories. Although the memory requirements of the AIDS facility are not excessive with respect to the TELLER program, they can be restrictive on some systems. For example, on-line aids like those of the TELLER system certainly cannot be provided on a 'personal' computer without some form of rapid-access mass storage (e.g., a floppy disk).

The effort required to incorporate on-line aids was reflected, in part, by the time spent on different implementation tasks. Approximately 61 hours were devoted to designing, coding, and debugging the programs comprising the TELLER system. Because the AIDS facility allows programs to be developed independently of specifying the <u>content</u> of on-line aids, very few messages were composed in conjunction with program development and testing.

Approximately 33 hours were devoted to composing, entering, and revising all messages: prompts, error messages, format descriptions, etc.

The flexibility of the AIDS facility was further demonstrated by the ease with which the rudimentary HELP mode was implemented. Program changes to provide the HELP mode required one hour; revising the messages required an additional hour.

Use of the AIDS facility revealed two limitations. First, references to the aids themselves in the messages were fixed. For example, several messages ended with the line

For more information, enter ?FMT again.

When the HELP mode was implemented, all these messages had to be changed. This could have been accomplished through script transfers or script parameters, but a more general feature should have been provided to display invocation codes known to the AIDS facility. A second limitation concerned the inter-relationship of aids as they were invoked. Because many aids shared the same messages, but were independent of one another, a user could have obtained the same message twice by invoking two seemingly different aids. No one actually encountered this situation in the experiments. However, the AIDS facility could eliminate redundant messages if the message for one aid were allowed to affect the status of another aid.

One final characteristic of the AIDS facility bears mentioning. The implied presence of the on-line aids during program implementation resulted in greater attention on the programmer's part to possible errors, their causes, and the content of messages in general. For example, in constructing lengthy explanations for some errors, inconsistencies in the command formats were more readily identified and, in some cases, corrected. The content of messages was also affected by a consideration of their underlying explanations. In many cases, it was found that 'lower-level' information could be included in an initial prompt or error message without unduly lengthening that message. These and other observations led to a set of guidelines for message composition, shown in Appendix J.

6.5 Conclusions

The pilot study demonstrated that for completely inexperienced users, the provision of on-line aids can adversely affect ease of use. It was observed that 1) users depended heavily on the manual, 2) users were reluctant to explicitly request on-line assistance, and 3) the presence of on-line aids confused users, resulting in poor performance. The following conclusions were derived by a one-way analysis of variance:

1) Differences in subjects' confidence that they could use the system without a manual were due to treatment effects ($p<.05$). The provision of

on-line aids (or the poor performance that resulted) caused subjects to have less confidence.

2) Differences in perceived ease of use could not be attributed to treatment effects.

The ensuing experiment assessed the influence of different types of assistance on ease of use by experienced programmers. The results, summarized in Table 6-9, indicated no significant differences due to treatment effects for overall perceived ease of use (TQS), computing costs incurred (COST), or total number of errors (TOT). On-line aids *were* found to significantly affect users' confidence that they could use the system without a manual (Q8, $p<.01$). Subjects who used the on-line aids required significantly less time (CNCT) to perform tasks than those who had no aids and those who had access to a HELP mode but did not use it ($p<.05$). The provision and use of on-line aids resulted in subjects making more unique errors ($p<.05$), but fewer repetitions of those errors ($p<.05$).

The provision of on-line aids through the AIDS facility was found to be straightforward from a programming standpoint. The facility also proved to be an effective tool for experimenting with on-line aids: usage was automatically monitored by the facility, and the nature of the aids was easily varied. However, the composition of messages required considerable effort. The version of the AIDS facility used in the experiment exhibited only one

Test	Between	Q8	TQS	COST	CNCT	TOT	UNQ	RPT
One-way ANOVA	NA:H:OA	.01	--	--	.05	--	--	.05
HSD	NA:H	--			--			--
	NA:OA	.01			.05			--
	H:OA	.01			--			.05
One-way ANOVA	NA':OA	.01	--	--	.05	--	.05	.05

Table 6-9. Summary of significant differences

limitation: the inability of an aid's invocation to affect the status of another aid. This limitation has been removed in a subsequent implementation. Finally, use of the AIDS facility helped to identify inconsistencies in the language and protocols, and resulted in clearer prompts and error messages.

7. Summary and Future Research

7.1 Summary

This dissertation has examined features and capabilities that comprise a user-oriented system. User-oriented features of a system are those supportive capabilities that improve user performance and satisfaction by 1) reducing the amount of data communicated between a user and a system for a given task, 2) improving a user's understanding of a system, 3) reducing the amount of information that must be remembered by a user during interaction, 4) minimizing the likelihood of user errors, and 5) enabling users with widely differing levels of experience to use the same system.

Four categories of user-oriented capabilities were identified: on-line aids, alternative communication modes, language extensibility, and defaults. Existing systems do not provide all of these forms of assistance consistently, for all users, and in a manner that facilitates their incorporation into programs. A consistent and suitably general set of capabilities was described for each of the identified categories of user aids. To provide these capabilities, a software user interface (UI) was designed, and portions thereof were implemented on a UNIVAC 1110.

The on-line aids provided by the UI are unobtrusive, situation-dependent, function-specific, and multi-level. The relationship between these properties and ease of use was investigated in an experiment. In a pilot study with inexperienced programmers, four groups were tested: a group with no on-line aids available, a group that had access to a rudimentary HELP mode, a group that had on-line aids with the properties under study, and a group that had the same on-line aids as the third group but was required to work without a manual. No significant differences were observed among the four treatments with respect to overall perceived ease of use. However, the on-line aids were observed to have a detrimental effect on user performance. In addition, subjects exposed to the on-line aids were significantly less confident than other subjects that they could use the system without a manual. The pilot study also revealed that inexperienced users were very dependent on written documentation and were reluctant to use the on-line aids. Other studies have also observed that facilities aimed at assisting users or improving productivity often go unused [BO74a, MA75b, SO79].

A subsequent experiment was conducted with more experienced programmers. On-line assistance and the nature of that assistance were not found to have a significant effect on overall perceived ease of use. However, significant differences between treatments were observed in several other measures of ease of use. Subjects who used the on-line aids were more confident that they could use the system without a manual, and required significantly less time to perform tasks. The provision and use of on-line aids resulted in subjects making more unique errors, but repeating those errors less often.

7.2 Future research

Several areas related to user-oriented capabilities need further study. The first area concerns the relationship between user aids and user experience. On-line aids appear to be especially useful to the experienced but infrequent user, or the user who is trying a system for the first time but has had experience with other systems. Empirical studies have lacked a well-defined notion of user experience. A more formal categorization of types of experience would make it possible to study what aids are best suited for a given user population. Some experience-related concepts that need analysis and further study are: measures of programming experience, measures of interactive experience, classifications of usage frequency, and concepts that are transferable from one system or language to another. The experiment described in this dissertation demonstrated the usefulness of the UI for studying such questions. The integration of user aids in a front-end facility makes it possible to easily vary their properties for purposes of a controlled study.

More generally, it is necessary to study user characteristics that that can be used to assist the user in an 'intelligent' manner. Some research in this area has already been conducted [HE74a]. The aids described in this dissertation are, for the most part, passive. Their effectiveness depends on the user's ability to recognize what type of assistance is required, and his willingness to use those assisting features. However, the study of user behavior in such a passive environment is a necessary first step in determining what forms of assistance are needed and how they can best be provided. For example, a tentative conclusion of this research is that inexperienced users require more system-directed assistance than experienced users. An extension of the aids described here would allow a host system to vary the form and content of communication according to the user profiles maintained by the UI.

A third area needing study is the degree to which user aids can be isolated from a program. As more systems rely on parsing techniques to interpret their command languages, it will be convenient to embed aid specifications in the grammars of these languages. It is not unreasonable to expect that a grammar could contain a specification similar to

```
  <column> ::= integer [COLUMN/QCOL/ECOL/DCOL]
```

where COLUMN is the name usable for keyword parameter specification, and QCOL, ECOL, and DCOL identify scripts used for prompting, error messages, and references to documentation, respectively. Such capabilities are of course limited by the nature of semantic errors. Furthermore, the more removed aids are from a program, the harder it is to make those aids situation-dependent. These conflicting goals are reflected in the attempts of some researchers to make dialogue specifications less procedural [BL77, HE74b], while others have proposed embedding those specifications in a programming language [LA78].

Finally, it is necessary to explore more effective methods for composing and organizing assisting messages and integrating them with written forms of documentation. The development of on-line aids will undoubtedly exhibit the same limitation encountered by proponents of computer-assisted instruction. Although the general software for each can be readily provided, the effort required to author the 'databases' for particular applications is considerable. Already, the maintenance of on-line documentation requires full-time personnel at several installations. Robertson, Newell, and Ramakrishna report that the 35,000 frame library for PROMIS required about 70 man-years to generate [RO77]. Such cost factors strongly suggest the need for automated assistance in constructing and maintaining on-line aids. In some cases, much of the material that goes into on-line aids duplicates that which is already available for computer-generated documentation. The THUMB system proposed by Price [PR78] is suggestive of how some of these materials and tasks can be integrated.

References

[AB75] Abrams, M. D., and Cotton, I. W., "The Service Concept Applied to Computer Networks", National Bureau of Standards Technical Note 880, August, 1975.

[AD69] Adams, J., and Cohen, L., "Time-sharing vs. Batch Processing", in Computers and Automation, Vol. 18, March, 1969, pp. 30-34.

[AS77] Ash, W., Bobrow, R., Grignetti, M., and Hartley, A., "Intelligent On-line Assistant and Tutor System", Technical Report No. 3607, Bolt Beranek and Newman, Inc., January, 1977.

[BE72] Bennett, J. L., "The User Interface in Interactive Systems", in Annual Review of Information Science and Technology, Volume 7, (ed. C. A. Cuadra), 1972, pp. 159-196.

[BE70] Bennick, F., and Frye, C. H., "PLANIT Reference Manual", CDC No. 97401700, October, 1970.

[BL77] Black, J. L., "A General Purpose Dialogue Processor", in Proceedings, 1977 National Computer Conference, pp. 397-408.

[BO75] Boehm, B. W., McClean, R. K., and Ufrig, D. B., "Some Experience with Automated Aids to the Design of Large-Scale Reliable Software", in Proceedings, International Conference on Reliable Software, 1975, pp. 105-113.

[BO74a] Boies, S. J., "User Behavior on an Interactive Computer System", IBM Systems Journal, Vol. 13, No. 1, 1974, pp. 2-18.

[BO74b] Boies, S. J., and Gould, J. D., "Syntactic Errors in Computer Programming", in Human Factors, Vol. 16, No. 3, June, 1974, pp. 253-257.

[BU78] Burton, R. R., and J. S. Brown, "An Investigation of Computer Coaching for Informal Learning Activities," Technical Report No. 3914, Bolt, Beranek, and Newman, Inc., August, 1978.

[CA72] Carlisle, J. H., "Interactive Man-Machine Communication," NTIS Document No. AD-740 101, March, 1972.

[CD73] Control Data Corporation, "CYBERNET/KRONOS 2.1 Time-Sharing Tutorial", Publication No. 86615800, 1973.

[CO57] Cochran, W. G., and Cox, G., Experimental Designs, Second Edition, John Wiley and Sons, Inc., 1957.

[CO76] Cohen, S., and Pieper, S. C., "The SPEAKEASY-3 Reference Manual", Argonne National Laboratory, May, 1976.

[CO77] Cohen, S., and Pieper, S. C., "The SPEAKEASY HELP Documents", Argonne National Laboratory, January, 1977.

[CO75] Cooke, J. E., and Bunt, R. B., "Human Error in Programming: The Need to Study the Individual Programmer", University of Saskatchewan Department of Computational Science Technical Report 75-3, February, 1975.

[CO62] Coulson, J. (ed.), Programmed Learning and Computer Based Instruction, John Wiley and Sons, 1962.

[DO70] Dolotta, T. A., "Functional Specifications for Typewriter-like Time Sharing Terminals", in Computing Surveys, Vol. 2, No. 1, March, 1970, pp. 5-31.

[DZ78] Dzida, W., Herda, S., and Itzfeldt, W. D., "User Perceived Quality of Interactive Systems", in Proceedings of the Third International Conference on Software Engineering, August, 1978, pp. 188-195.

[EN75] Endres, A., "An Analysis of Errors and Their Causes in System Programs", in Proceedings, International Conference on Reliable Software, 1975, pp. 327-336

[FD76] A Progress Report on the Activities of the CODASYL End-User Facility Task Group, FDT Bulletin of ACM SIGMOD, Vol. 8, No. 1, 1976.

[FI74] Fife, D. W., "Research Considerations in Computer Networking to Expand Resource Sharing", National Bureau of Standards Technical Note 801, June, 1974.

[GA75] Gannon, J. D., and Horning, J. J., "Language Design for Programming Reliability", in IEEE Transactions on Software Engineering, Vol. SE-1, No. 2, June, 1975, pp. 179-191. (also in Proceedings, International Conference on Reliable Software, 1975, pp. 10-22).

[GA76] Gannon, J. D., "An Experiment for the Evaluation of Language Features", in International Journal of Man-Machine Studies, Vol. 8, 1976, pp. 61-73.

[GA77] Gannon, J. D., "An Experimental Evaluation of Data Type Conventions", in Communications of the ACM, Vol. 20, No. 8, August, 1977, pp. 584-595.

[GI77] Gilb, T., and Weinberg, G. M., Humanized Input, Winthrop Publishers, Inc., 1977.

[GL72] Glantz, R. S., "Design of an On-Line Query Language for Full Text Patent Search", Report M72-97, Mitre Corporation, December, 1972.

[GO74] Gould, J. D., and Drongowski, P., "An Exploratory Study of Computer Program Debugging", in Human Factors, Vol. 16, No. 3, June, 1974, pp. 258-277.

[GO75a] Gould, J. D., and Ascher, R., "Use of an IQF-like Query Language by Non-programmers", IBM Research Report RC 5279, February 20, 1975.

[GO75b] Gould, J. D., Lewis, C., and Ascher, R., "Some Psychological Evidence on How People Debug Computer Programs", in International Journal of Man-Machine Studies, Vol. 7, No. 2, 1975, pp. 151-182.

[GR67] Grant, E. E., and Sackman, H., "An Exploratory Investigation of Programmer Performance Under On-line and Off-line Conditions", in IEEE Transactions on Human Factors, Vol. HFE-8, 1967.

[GR74] Greenberger, M., et al (eds.). "Networks for Research and Education: Sharing Computer and Information Resources Nationwide", MIT press, 1974.

[GR76] Grignetti, M. C., Ash, W. L., Bobrow, R. J., Gould, L., and Hartley, A. K., "Intelligent On-Line Assistant and Tutor System", Technical Report Number 3365, Bolt Beranek and Newman, Inc., June, 1976.

[HA72] Hagerty, P. E., "The University of Maryland DUM (Demand User's Monitor) System", in Proceedings, USE Fall Conference, 1972, pp. 29-80.

[HA73a] Halstead, M. H., "Language Selection for Applications", in Proceedings, 1973 National Computer Conference, pp. 211-214.

[HA76] Hansen, J. V., "Man-Machine Communication: An Experimental Analysis of Heuristic Problem-Solving Under On-Line and Batch-Processing Conditions", in IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-6, No. 11, November, 1976, pp. 746-752.

[HA71] Hansen, W. J., "User Engineering Principles for Interactive Systems", in Proceedings, 1971 Fall Joint Computer Conference, pp. 523-532.

[HA73b] Hays, W. L., Statistics for the Social Sciences, Second Edition, Holt, Rinehart, and Winston, Inc., 1973.

[HE74a] Heafner, J. F., "A Methodology for Selecting and Refining Man-Computer Languages to Improve Users' Performance", University of Southern California Technical Report ISI/RR-74-21, September, 1974.

[HE74b] Heindel, L. E., and Roberto, J. T., "LANG-PAK: An Interactive Language Design System", American-Elsevier Series Monograph, March, 1974.

[HO73] Hoare, C. A. R., "Hints on Programming Language Design", Stanford University Computer Science Department Technical Report CS-403, December, 1973.

[JP75] "Introduction to JPLDIS", Jet Propulsion Laboratory, California Institute of Technology, June, 1975.

[KA75] Kamman, R., "The Comprehensibility of Printed Instructions and the Flowchart Alternative", in Human Factors, Vol. 17, No. 2, April, 1975, pp. 183-191.

[KE74] Kennedy, T. C. S., "The Design of Interactive Procedures for Man-Machine Communication", in International Journal of Man-Machine Studies, Vol. 6, No. 3, May, 1974, pp. 309-334.

[KI68] Kirk, R. E., Experimental Design: Procedures for the Behavioral Sciences, Wadsworth Publishing Co., 1968.

[KL73] Kling, R., "Who Needs a Person-Centered Computer Technology?", University of Wisconsin Computer Science Department Technical Report 169, July, 1973.

[KN71] Knuth, D. E., "An Empirical Study of FORTRAN Programs", in Software Practice and Experience, Vol. 1, No. 2, April-June, 1977, pp. 105-133.

[KR76] Kriloff, H. Z., "Human Factor Considerations for Interactive Display Systems", in User-Oriented Design of Interactive Graphics Systems (ed. S. Treu), ACM/SIGGRAPH Workshop, October, 1976.

[LA78] Lafuente, J. M., and Gries, D., "Language Facilities for Programming User-Computer Dialogues", IBM Journal of Research and Development, Vol. 22, No. 2, March, 1978, pp. 145-158.

[LI60] Licklider, J. C. R., "Man-Computer Symbiosis", in IRE (IEEE) Transactions on Human Factors in Electronics, Vol. 1, No. 1, March, 1960.

[LI76] Litecky, C. R., and Davis, G. B., "A Study of Errors, Error-Proneness, and Error Diagnosis in COBOL", in Communications of the ACM, Vol. 19, No. 1, January, 1976, pp. 33-37.

[LO77a] Lochovsky, F. H., Data Base Management Systems User Performance, Ph. D. Thesis, University of Toronto, 1977.

[LO77b] Lochovsky, F. H., and Tsichritzis, "User Performance Considerations in DBMS Selection", in Proceedings, SIGMOD International Conference on Management of Data, 1977, pp. 128-134.

[LO77c] Love, T., "An Experimental Investigation of the Effect of Program Structure on Program Understanding", in Proceedings of the ACM Conference on Language Design for Reliable Software, March, 1977, pp. 105-113.

[MA75a] Mann, W. C., "Why Things Are So Bad for the Computer-Naive User", in Proceedings, 1975 National Computer Conference, pp. 785-787.

[MA76] Marcus, S., and Reintjes, J. F., "The Networking of Interactive Bibliographic Retrieval Systems", MIT Electronic Systems Laboratory, March, 1976.

[MA73] Martin, J., Design of Man-Computer Dialogues, Prentice Hall, 1973.

[MA75b] Martin, T. H., and Parker, E. B., "Comparative Analysis of Interactive Retrieval Systems", in SIGPLAN Notices, Vol. 10, No. 1, January, 1975, pp. 75-85.

[MI74] Miller, L. A., "Programming by Non-Programmers", in International Journal of Man-Machine Studies, Vol. 6, No. 2, March, 1974, pp. 237-260.

[MI75] Miller, L. A., "Naive Programmer Problems with Specification of Transfer of Control", in Proceedings, 1975 National Computer Conference, pp. 657-663.

[MI77] Miller, L. H., "A Study in Man-Machine Interaction", in Proceedings, 1977 National Computer Conference, pp. 409-421.

[MI68] Miller, R. B., "Response Time in Man-Computer Conversational Transactions", in Proceedings, 1968 Fall Joint Computer Conference, pp. 267-277.

[MO79] Moher, T., and Schneider, G. M., "Methodology for Experimental Research in Software Engineering", University of Minnesota Computer Science Department, Technical Report 79-2, January, 1979.

[NE73] Neumann, A. J., "Network User Information Support", National Bureau of Standards Technical Note 802, December, 1973.

[NI68] Nickerson, R. S., Elkind, J. I., and Carbonell, J. R., "Human Factors and the Design of Time-sharing Computer Systems", in Human Factors, Vol. 10, No. 2, March, 1968, pp. 127-134.

[PO77] Poorbaugh, H. J., "OLDS: An On-Line Documentation System", in Proceedings, ACM SIGUCC User Services Conference, Kansas City, 1977.

[PR74] Pradels, J. L., "The Guide: An Information System", University of Illinois at Urbana-Champaign Department of Computer Science, Technical Report UIUCDCS-R-74-626, March, 1974.

[PR78] Price, L. A., Representing Text Structure for Automatic Processing, Ph. D. Thesis, University of Wisconsin, May 1978, (also available as University of Wisconsin Computer Science Department Technical Report No. 324).

[PU73] Puerling, B. W., and Roberto, J. T., "The Natural Dialogue System", Bell System Technical Journal, Vol. 52, No. 10, December, 1973, pp. 1725-1742.

[RA72] Raub, W. F., "Automated Information Handling in Pharmacology Research", in Proceedings, 1972 Spring Joint Computer Conference, pp. 1160-1161.

[RE75] Reisner, P., Boyce, R. F., and Chamberlin, D. D., "Human Factors Evaluation of Two Data Base Query Languages -- SQUARE and SEQUEL", in Proceedings, 1975 National Computer Conference, pp. 447-452.

[RE77] Reisner, P., "Use of Psychological Experimentation as an Aid to Development of a Query Language", in IEEE Transactions on Software Engineering, Vol. SE-3, No. 3, May, 1977, pp. 218-229.

[RO70] Roberts, R., "HELP -- A Question Answering System", in Proceedings, 1970 Fall Joint Computer Conference, pp. 547-554.

[RO77] Robertson, G., Newell, A., and Ramakrishna, K., "ZOG: A Man-Machine Communication Philosophy", Carnegie-Mellon University Department of Computer Science, Interim Report, August, 1977.

[RO67] Root, R. T., and Sadacca, R., "Man-Computer Communication Techniques: Two Experiments", in Human Factors, Vol. 9, No. 6, December, 1967, pp. 521-528.

[SA70] Sackman, H., "Experimental Analysis of Man-Computer Problem-Solving", in Human Factors, Vol. 12, No. 2, 1970, pp. 187-201.

[SA73] Sackman, H., "Some Exploratory Experience with Easy Computer Systems", in Proceedings, 1973 National Computer Conference, pp. M30-M33.

[SC67] Schatzoff, M., Tsao, R., and Wiig, R., "An Experimental Comparison of Time-sharing and Batch Processing", in Communications of the ACM, Vol. 10, May, 1967, pp. 261-272.

[SH75a] Shapiro, S. C., and S. C. Kwasny, "Interactive Consulting Via Natural Language," in Communications of the ACM, Vol. 18, No. 8, August, 1975, pp. 459-462.

[SH74] Shneiderman, B., "Experimental Testing in Programming Languages, Stylistic Considerations and Design Techniques", in _Proceedings, 1975 National Computer Conference_, pp. 653-656 (also available as University of Indiana Computer Science Department Technical Report No. 16).

[SH75b] Shneiderman, B., and Mayer, R., "Towards a Cognitive Model of Programmer Behavior", University of Indiana Computer Science Department Technical Report No. 37, August, 1975.

[SH76a] Shneiderman, B., and McKay, D., "Experimental Investigations of Computer Program Debugging and Modification", _Proceedings of the Sixth International Congress of the International Ergonomics Association_, July, 1976.

[SH76b] Shneiderman, B., "Exploratory Experiments in Programmer Behavior", in International Journal of Computer and Information Sciences, Vol. 5, June, 1976, pp. 123-143 (also available as University of Indiana Computer Science Department Technical Report No. 17).

[SH77a] Shneiderman, B., "Improving the Human Factors Aspect of Database Interactions", University of Maryland Department of Information Systems Management Technical Report No. 26, September, 1977.

[SH77b] Shneiderman, B., "Measuring Computer Program Quality and Comprehension", International Journal of Man-Machine Studies, Vol. 9, 1977, pp. 1-14.

[SH77c] Shneiderman, B., Mayer, R., McKay, D., and Heller, P., "Experimental Investigations of the Utility of Detailed Flowcharts in Programming", in Communications of the ACM, Vol. 20, No. 6, June, 1977, pp. 373-381.

[SI73] Sime, M. E., Green, T. R., and Guest, D. J., "Psychological Evaluation of Two Conditional Constructs Used in Computer Languages", in International Journal of Man-Machine Studies, Vol. 5, No. 1, 1973, pp. 105-113.

[SM67] Smith, L. B., "A Comparison of Batch Processing and Instant Turnaround", in Communications of the ACM, Vol. 10, August, 1967, pp. 495-500.

[SO79] Sondheimer, N. K., "On the Fate of Software Enhancements", in Proceedings, 1979 National Computer Conference, pp. 1043-1049.

[ST74] Sterling, T. D., "Guidelines for Humanizing Computerized Information Systems: A Report from Stanley House", in Communications of the ACM. Vol. 17, No. 11, November, 1974, pp. 609-613.

[TA75] Taulbee, O. E., Treu, S., and Nehnevajsa, J., "User Orientation in Networking", in Proceedings, 1975 National Computer Conference, pp. 637-644.

[TE72] Teitelman, W., "Do What I Mean: The Programmer's Assistant", in Computers and Automation, Vol. 21, No. 4, April 1972, pp. 8-11.

[TE78] Teitelman, W., "A Display Oriented Programmer's Assistant", in Proceedings, Fifth International Joint Conference on Artificial Intelligence, Vol. 2, August, 1977, pp. 905-915.

[TH75] Thomas, J. C., and Gould, J. D., "A Psychological Study of Query by Example", in Proceedings, 1975 National Computer Conference, pp. 439-445

[TR75] Treu, S., "Interactive Command Language Design Based on Required Mental Work", in International Journal of Man-Machine Studies, Vol. 7, No. 1, 1975, pp. 135-149.

[TR77] Truitt, T. D., and Emery, J. C., "Provision of User Services in a Network Environment," ACM SIGUCC Newsletter, Vol. 7, No. 1, pp. 5-13.

[UN72] UNIVAC, division of Sperry Rand, "Conversational Time Sharing (CTS) System, Programmer's Reference Manual", UP-7940, 1972.

[VE77a] Venezky, R. L., Relles, N., and Price, L. A., "Man-Machine Integration in a Lexical Processing System", in Cahiers de Lexicologie, Vol. 30, 1977, pp. 17-46.

[VE77b] Venezky, R. L., Relles, N., and Price, L. A., "User Aids in a Lexical Processing System", in Proceedings of the Third International Conference on Computing in the Humanities (eds. J. S. North and S. Lusignan), August, 1977, pp. 317-325.

[VE77c] Venezky, R. L., Relles, N., and Price, L. A., "LEXICO: A System for Lexicographic Processing", in Computers and the Humanities, Vol. 11, 1977, pp. 127-137.

[WA72] Warshall, S., "AMBUSH - A Case History in Language Design", Proceedings, 1972 Spring Joint Computer Conference, pp. 321-332.

[WA74] Walther, G., and O'Neil, H., "On-line User-Computer Interface - The Effects of Interface Flexibility, Terminal Type, and Experience on Performance", in Proceedings, 1974 National Computer Conference, pp. 379-384.

[WE71] Weinberg, G. M., The Psychology of Computer Programming, 1971, Van Nostrand Reinhold Publishing Company.

[WE74a] Weissman, L., "Psychological Complexity of Computer Programs: An Initial Experiment", University of Toronto Technical Report CSRG-26.

[WE74b] Weissman, L., A Methodology for Studying the Psychological Complexity of Computer Programs, Ph. D. Thesis, University of Toronto, 1974.

[YO74] Youngs, E. A., "Human Errors in Programming", in International Journal of Man-Machine Studies, Vol. 6, No. 3, May, 1974, pp. 361-376.

Appendix A

This appendix provides an overview of the TELLER system, used in the experiment described in chapter 6. This overview is taken from the first chapter of the TELLER manual given to subjects.

1.1 Overview

TELLER is an interactive computer system for maintaining several bank accounts. The system allows you to establish new accounts, enter information about those accounts (e.g., deposits and withdrawals), and receive information about any account (e.g., periodic statements). You can maintain any number of accounts representing actual accounts at any number of banks. Each account is a savings, checking, or mortgage (loan) account. For example, you might use the TELLER system to maintain information about

> two savings accounts and one checking account at bank X,
>
> a savings account and a loan account at bank Y, and
>
> a savings account and checking account at bank Z.

By maintaining a record of your accounts through the TELLER system, you can check your bank's monthly statements, keep an up-to-date record of your accounts' balances, or perhaps analyze your banking practices. To begin using the TELLER system, you must enter

```
@TELLER
```

at your terminal. You may then enter commands to perform any of the following tasks:

1) open an account of any type (savings, checking, or loan),
2) review an existing account -- adding new entries, changing entries, or requesting summary statements,
3) close out an account, and
4) display a summary of your accounts maintained by the TELLER system.

1.2 Opening an account

When you enter the OPEN command, the system displays several questions. For example, when opening a checking account, the following questions are displayed, to which you would respond as indicated:

TELLER system displays	You should enter
bank name?	the name of the bank at which you are opening the new account
acct# ?	the account number of the newly opened account
depositor?	the name(s) of the account's owner(s)
date opened?	the date that the account was opened
minimum?	the minimum balance that must be maintained in the account without incurring a service charge
check charge?	the amount charged by the bank for each check written

The following interaction illustrates how a new checking account is opened. Commands and data entered by the user are underlined.

```
@TELLER
TELLER system;  09/10/78   10:35

Task?  OPEN C

bank name?  FIDELITY NATIONAL BANK
acct# ?  1357-239
depositor?  BILL AND MARY
date opened?  09/01/78
minimum?  100
check charge?   0

Account successfully opened.
```

1.3 Reviewing an existing account

Any existing account may be reviewed for a number of purposes, including

a) adding new entries (e.g., deposits and withdrawals)
b) changing entries (e.g., changing the amount of a check), or
c) displaying information regarding the account (e.g., a monthly statement).

To begin reviewing an account, you can enter one of the following task commands:

```
SEE C to access a checking account
SEE L to access a loan account
SEE S to access a savings account
```

If you enter just the command SEE, the system will ask you for the type of account to be accessed. If you have more than one such account, the system will ask you to identify the account you want to SEE.

1.4 Transactions

After you have initiated an OPEN or SEE task, you may enter commands to view or change the designated account. These commands are called transactions, e.g.: depositing some amount in a savings account, changing the amount of a check, or requesting the monthly statement for an account. For example, to record a deposit, the following transaction command might be entered:

```
DEPOSIT  10/15/77  $100.00
```

Transactions are described in detail in chapter 3. After all transactions have been entered for an account, a task is terminated by entering the command

```
END
```

Another task may then be initiated (e.g., reviewing another account).

1.5 Obtaining directories

The DIRECTORY command allows you to obtain a summary of your TELLER system accounts. Several types of directories may be requested. For example, by entering

```
     DIRECTORY ALL
```

you can obtain a complete listing of all your accounts, grouped by their type (first checking accounts, then loan accounts, and finally savings accounts). Within each group, accounts will be listed in alphabetical order by bank names. An example of such a directory appears below.

```
*** CHECKING ACCOUNTS ***

BANK NAME                ACCOUNT #   DEPOSITOR     BALANCE
------------------------ ----------  -----------  ---------
SECOND NATIONAL BANK     201-85668   LEE         -$   87.20
SECOND NATIONAL BANK     201-85665   TERRY        $   86.60
SECURITY SAVINGS & LOAN  1-96-222    TERRY        $  306.54
                                          (TOTAL: $  305.94)

*** LOAN ACCOUNTS ***

BANK NAME                ACCOUNT #   DEPOSITOR     BALANCE
------------------------ ----------  -----------  ---------
UNION TRUST BANK         369-2835    LEE          $  443.00
UNION TRUST BANK         369-5113    LEE & TERRY  $26579.00
                                          (TOTAL: $27022.00)

*** SAVINGS ACCOUNTS ***

BANK NAME                ACCOUNT #   DEPOSITOR     BALANCE
------------------------ ----------  -----------  ---------
UNION TRUST BANK         520-3114    TERRY        $ 1525.00
SECOND NATIONAL BANK     839-01456   LEE & TERRY  $  525.00
SECURITY SAVINGS & LOAN  1-05-339    JENNIFER     $  200.00
SECURITY SAVINGS & LOAN  1-09-524    TOMMY        $  332.85
                                          (TOTAL: $ 2582.85)
```

1.6 Terminating a session

After all accounts have been accessed, you should end the session by entering the command

```
     QUIT
```

However, if you have made many mistakes, or if you want to rescind all your tasks for any other reason, you may enter the command

```
FORGET
```

The FORGET command rescinds everything you did after starting the session. All your accounts will appear exactly as they did at the start of the session (when you originally entered @TELLER).

1.7 Summary

Three types of accounts are maintained by the TELLER system: savings, checking, and loan accounts. A session consists of several tasks -- e.g., opening an account, viewing accounts, and displaying directories. Each task is initiated by entering a task command when the system displays

```
Task?
```

The OPEN and SEE tasks consist of individual transactions, and must be terminated by an END command before proceeding to the next task. When all tasks have been completed, a session is terminated by either a QUIT or FORGET command. The following example of a session illustrates several tasks within a session. Commands and data entered by the user are underlined.

```
@TELLER
TELLER system;  09/10/78   10:35

Task? DIRECTORY ALL

   [system displays directory of all accounts]

Task? OPEN S

   transaction
   transaction
        .          [transactions are entered
        .           to open new savings account]
        .
   END

Task? SEE C PRUDENTIAL S&L

   transaction
   transaction
        .          [checking account is updated;
        .           new checks and deposits are
        .           recorded, monthly statement
   END                     printed, etc.]

Task? SEE L

   transaction
   transaction
        .                 [loan account is reviewed, deposits
        .                  recorded, etc.]
        .
   END

Task? QUIT                [The session is ended; all accounts
                           are updated according to the tasks
                           performed in this session.]
```

1.8 Command formats

Some task and transaction commands are followed by information such as a date, a dollar amount, or a bank name. Any number of blanks (but at least one) may separate parts of a command. For example, all of the following commands are equivalent:

```
SEE C 159-334
SEE    C   159-334
SEE C     159-334
SEE    C 159-334
```

The following rules must also be followed regarding parts of a command:

. Blanks must not be included as part of an account number, a date, a dollar amount, or a percentage (i.e., interest).

. Dates are specified in the form
   MM/DD/YY
where MM is the number of the month (1 for January, 2 for February, etc.), DD is the day, and YY is two digits representing the year (e.g., 78 for 1978). All three portions must be specified, and they must be separated by a slash (/). Blanks are not allowed within a date; however, for days and months less than 10, a preceding zero is optional. Thus, the following date formats are equivalent, all representing March 7, 1978:

```
3/7/78
03/7/78
3/07/78
```

03/07/78

. Dollar amounts are represented as a decimal number (e.g., 5.75) or integer (e.g., 100) optionally preceded by a dollar sign ($). Commas and blanks are not allowed within a dollar amount. Thus, the following are equivalent forms for representing one hundred dollars:

```
$100
$100.00
100
100.00
$100.
```

. Interest is expressed as a decimal number (e.g., 5.5) or an integer (e.g., 5). The percent sign (%) must not be used when representing interest.

. Bank names, depositors, and memoes may include blanks. However, two or more consecutive blanks are treated as a single blank.

Appendix B

This appendix describes the HELP mode and on-line aids used by the H and OA groups, respectively, in the experiment described in chapter 6. The descriptions are taken from the manuals given to subjects.

1.9 System aids (HELP mode)

The TELLER system can also provide information about its use. The HELP command may be entered to obtain information about task commands, transaction commands, their formats, and how they are used. The HELP command may be entered whenever the system is expecting a command, i.e., when any of the following questions is displayed:

```
Task?

transaction (D, W, C, R, S, or L) ?

transaction (D, W, I, C, R, or S) ?

transaction (P, S, C, or M) ?
```

By using the HELP command, you can minimize the need to refer to this manual. After you enter the HELP command, you will be in HELP mode. The system will ask you to enter the terms or commands for which you want information. Each time the system prompts you by displaying

```
-->
```

you may enter any of the following:

1. a command or term for which you want an explanation;
2. the word TERMS, for a list of valid commands and terms that the system can explain;
3. a blank line (just pressing the RETURN key) for further explanation of the material just displayed;
4. the word END, indicating that you have obtained the desired information and want to return to the TELLER system.

When you have obtained information for all the commands and terms of interest to you, enter END. You will thereby leave HELP mode and return to the TELLER system. You can switch to HELP mode as often as you want.

1.9 System aids (on-line aids)

The TELLER system can also provide information about its use. Several special keys may be used to obtain explanations of errors or questions, descriptions of commands, or examples of correct input. These keys may be used at any time and do not affect whatever task or transaction you are in the midst of performing.

By using the special requests for assistance described below, you can minimize the need to refer to this manual. You can enter any of the requests any time you want and as many times as you want.

?QUESTION: Whenever the system asks a question or is waiting for you to supply a command or data, if you do not understand the question, press the key marked EXPLAIN QUESTION, or enter

```
?QUESTION
```

This entry represents a request to explain the question. If you still don't understand the question, you may enter ?QUESTION again, and do this repeatedly to obtain successively more detailed explanations and examples. You can also use ?QU in place of ?QUESTION. Use ?QUESTION or ?QU whenever you don't understand what you should enter next.

?ERROR: Whenever the system displays an error message, if you don't understand the error, press the key marked EXPLAIN ERROR, or enter

```
?ERROR
```

This entry represents a request to explain the error. If you still don't understand the error, you may enter ?ERROR again, and do this repeatedly to obtain successively more detailed explanations and examples. You can also use ?ERR in place of ?ERROR. Use ?ERROR or ?ERR whenever the system has displayed an error message you don't understand.

?EXAMPLE: Whenever the system displays an error message, if you would like examples of correct input, press the key marked EXAMPLE, or enter

```
?EXAMPLE
```

To obtain further examples, you may enter ?EXAMPLE as many times as you want. You can also use ?EG in place of ?EXAMPLE.

?FORMAT: Anytime you want to know the format of a command (or part of a command), you may press the key marked EXPLAIN FORMAT, or enter

```
?FORMAT
```

followed by the command or item whose format you are interested in. As always, after entering this aid, you may repeatedly enter it again to obtain successively more detailed descriptions. For example, to obtain a description of the SEE command, you may enter at any time

```
?FORMAT  SEE
```

Similarly, to obtain a description of the date portion of commands, you may enter at any time

```
?FORMAT  DATE
```

You can use ?FMT in place of ?FORMAT. Any number of blanks may be used between ?FORMAT (or ?FMT) and the command or item of interest. The ?FORMAT request is especially useful when you know what command to use, but aren't sure of its format.

You can also enter ?FORMAT (or ?FMT) by itself when the system displays an error message. This is a way of asking the system "What is the correct format of the command or data just entered incorrectly?"

?HELP: To obtain a description of all these aids and their use, press the key marked HELP, or enter

```
?HELP
```

any time. You can enter this repeatedly to obtain successively more detailed descriptions of all the aids available.

Appendix C

This appendix contains the questionnaire filled in by subjects in the pilot study and the experiment. Subjects in the NA (no aids) group were given this questionnaire without questions Q11 and Q12. Questions Q11 and Q12 differed for the H (HELP mode) and OA (on-line aids) groups, as indicated by the bracketed portions.

Please answer the following questions. Under each question, circle the number that corresponds to your answer.

(Q1) How easy is it to choose which command to use in a given situation?

1. very hard to choose right command
2. most commands are hard to remember
3. some commands are hard to remember; others are easy
4. most commands are easy to remember
5. very easy to choose right command

(Q2) How easy is it to enter commands?

1. all commands are very difficult to enter correctly
2. most commands are difficult to enter correctly
3. some commands are difficult; others are easy
4. most commands are easy to enter correctly
5. all the commands are very easy to enter correctly

(Q3) How well did you understand questions asked by the system?

1. all the questions are very hard to understand
2. most questions are hard to understand
3. some questions are hard; others are easy
4. most questions are easy to understand
5. all questions are very easy to understand

(Q4) How well did you understand any errors you made?

1. all errors are hard to understand
2. most errors are hard to understand
3. some errors are hard; others are easy
4. most errors are easy to understand
5. all errors are very easy to understand

(Q5) How effective is the manual when <u>learning</u> how to use the system (before using the system)?

1. The manual is very confusing
2. Most of the manual is hard to understand
3. Some parts of the manual are confusing; other parts are clear.
4. Most of the manual is clear.
5. The manual is very clear and easy to understand.

(Q6) How useful is the manual <u>while</u> <u>using</u> the system?

1. The manual is very hard to use and understand.
2. Most of the manual is hard to use.
3. Some parts of the manual are hard to use; other parts are clear.
4. Most of the manual is clear.
5. The manual is very clear and useful.

(Q7) Can you use the system without someone to help you?

1. I can't use the system without constant help from someone.
2. I would need help frequently from someone.
3. I would need occasional help from someone.
4. I can get by with very little help from someone.
5. I can use the system without anyone's help.

(Q8) Can you use the system without referring to the manual?

1. I can't use the system without constantly referring to the manual.
2. I would need to refer to the manual frequently.
3. I would need to refer to the manual occasionally.
4. I can use the system with very little reference to the manual.
5. I can use the system without the manual.

(Q9) How reasonable is the time required to perform tasks?

1. Almost all tasks take too much time to complete correctly.
2. Most tasks take too long.
3. Some tasks take too long; others take a reasonable amount of time.
4. Most tasks take a reasonable amount of time.
5. Almost all tasks take a very reasonable amount of time.

(Q10) Overall, how hard is it to use the system?

1. The whole system is very hard to use.
2. Most of the system is hard to use.
3. Some parts of the system are hard to use; others are easy.
4. Most of the system is easy to use.
5. The whole system is very easy to use.

(Q11) Was the information obtained from the system (by entering ?HELP, ?ERROR, ?QUESTION, etc.) [by entering HELP]

1. no help at all
2. not so helpful
3. sometimes helpful
4. usually helpful
5. very helpful
6. I didn't use ?HELP, ?ERROR, ?QUESTION, etc. (Please explain briefly why you didn't use the on-line aids.)

   [I didn't use HELP. (Please explain briefly why you didn't use the HELP feature.)]

(Q12) How difficult is it to obtain assistance from the system (by entering ?HELP, ?ERROR, ?QUESTION, etc.) [by entering HELP]

1. on-line aids are [HELP mode is] very difficult to use
2. on-line aids are [HELP mode is] somewhat hard to use
3. on-line aids are [HELP mode is] sometimes hard to use, sometimes easy to use
4. on-line aids are [HELP mode is] usually easy to use
5. on-line aids are [HELP mode is] very easy to use

(Q13) Is anything in the manual incorrect or misleading? If so, please explain.

(Q14) Do you have any suggested improvements in the system? If so, what are they?

(Q15) Is anything displayed by the on-line aids (?HELP, ?ERROR, ?QU, etc.) [by HELP mode] incorrect or misleading? If so, what?

Appendix D

This appendix contains the instructions read to each subject before and after reading the manual. The same instructions were used in the pilot study and in the experiment.

"Please read this manual, which describes an interactive computer system for managing bank accounts. When you are finished, you will be asked to perform a few simple tasks using the system, as if you were managing a set of your own personal accounts. You will be able to use the manual as you perform these tasks.* You may take as much time as you like to read the manual, but try to keep it around 30 minutes, so that you have adequate time to perform all the tasks. Please tell me when you have finished reading the manual."

[subject reads manual]

"These pages [hand instructions to subject] describe a few simple tasks for the TELLER system; please perform the tasks in the same order that they appear. To perform the tasks, you may use any of the commands described in the manual. [H, OA, and NM subjects: You may also use any of the system aids described in the manual.]"

* In the pilot study, subjects in the NM (no manual) group were told that they would be asked to use the system without a manual.

Appendix E

This appendix describes the tasks given to subjects in the pilot study. Subjects in all four groups were given sheets with the following instructions.

> Remember that to transmit a line you must depress the RETURN key at the end of that line.
>
> If you make a mistake while entering a line, use the backspace key and enter the correct character, or use CTRL-X (hold down the CTRL key and press X) and re-enter the entire line.
>
> All dates in these instructions are for 1978.

1. Display a directory of all accounts in the system.

2. Fidelity Savings and Loan has changed its name to First Fidelity Savings. Change both accounts at this bank to reflect this name change. (Use the CHANGE task command.)

3. Close out Fran & Jerry's savings account at the Sic Semper Savings Bank.

4. Using the SHOW transaction, display a statement for the last month (September 1, 1978, through the end) in Jerry's checking account at the Sic Semper Savings Bank (account number 2-60-333).

   Close out the account (using the CLOSE task command.

5. Make the following changes to Fran's checking account at Prudential Bank & Trust:

   A deposit of one hundred dollars ($100), made on August 29th, was not recorded. Record the deposit.

   Check # 432, made out Aug. 20, should be in the amount of $25.40, rather than $24.50. Correct the entry (using the CORRECT transaction).

   Remove check number 427 ($85 for day care; 8/10/78). (Use the REMOVE transaction.)

Two checks were not recorded in the account; enter them now, using the WITHDRAW (or W) transaction. The checks are

421, August 3; $8 for gas
438, August 31; $15 for play tickets

List all checks for gas in August and September (between 8/1 and 9/30) using the LIST (or L) transaction.

A deposit made on August 20 was incorrectly recorded as $230. It should be for $250.00; make this correction.

6. Display a directory of all checking accounts.

7. Enter the following checks, deposits, and service charges in Jerry's checking account at Prudential Bank & Trust (account # 357-21001):

check no.	date	check issued to	check amount	deposit amount
863	9/5	newspaper	6.50	
864	9/12	ASL dues	17.00	
865	9/14	gas	5.00	
	9/15	service charge	.20	
866	9/15	watch repair	9.83	
	9/15	DEPOSIT		200.00

(Remember that you can use the CORRECT (C) transaction to correct any entries if you make a mistake.)

Display an account statement for the period covered by the entries you have just made. (Use the SHOW or S transaction.)

8. Open a savings account at Lake City Bank. The account number is 345-1101, interest is 6.5% compounded quarterly, and the account is to be in Fran's name. The account was opened on June 1, 1978. Record the following entries:

a deposit of $1,250 when the account was opened on June 1;

a deposit of $500.00 on August 15;

a withdrawal of 50 dollars on September 10;

a withdrawal of $630.00 on September 20;

a deposit of $200 on September 25;

a withdrawal of fifty dollars ($50) on October 15.

Display an account statement for the month of September (from 9/1 through 9/30) using the SHOW (or S) transaction.

9. Display a directory of all savings accounts.

10. Display a directory of all accounts at Sic Semper Savings.

Appendix F

The tables shown below summarize responses of pilot study subjects to the questionnaire.

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	TOS
	3	4	5	4	5	5	5	4	5	4	44
	4	4	4	3	4	5	4	4	4	4	40
NO	4	4	4	4	5	5	4	3	4	4	41
AIDS	4	4	5	4	4	5	5	2	4	4	41
	5	5	4	4	4	4	4	3	5	5	43
	4	5	5	4	5	5	5	3	4	5	45
mean	4.0	4.3	4.5	3.8	4.5	4.8	4.5	3.2	4.3	4.3	42.3
s.d.	.63	.52	.55	.41	.55	.41	.55	.75	.52	.52	1.97

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	TOS
	4	5	4	3	4	4	4	3	4	4	39
	5	5	5	5	5	5	5	2	5	5	47
HELP	4	4	3	3	4	4	4	3	4	4	37
MODE	4	4	5	5	5	5	5	3	4	4	44
	4	4	4	4	4	5	4	3	4	4	40
	2	5	3	3	3	3	2	2	3	2	28
mean	3.8	4.5	4.0	3.8	4.2	4.3	4.0	2.7	4.0	3.8	39.2
s.d.	.98	.55	.89	.98	.75	.82	1.1	.52	.63	.98	6.55

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	TOS
	3	5	4	2	5	4	2	1	1	4	31
	4	3	4	4	4	5	4	3	4	4	39
ON-LINE	4	5	4	4	4	5	5	3	5	5	44
AIDS	3	4	3	2	3	4	3	1	4	4	31
	4	4	5	4	4	5	4	1	3	4	38
	5	5	4	4	4	5	4	2	4	4	41
mean	3.8	4.3	4.0	3.3	4.0	4.7	3.7	1.8	3.5	4.2	37.3
s.d.	.75	.82	.63	1.0	.63	.52	1.0	.98	1.4	.41	5.32

Appendix G

This appendix describes the tasks given to subjects in the experiment. Subjects in all four groups were given sheets with the following instructions.

All dates in these instructions are for 1978.

1. Display a directory of all loan (L) accounts in the system.

   Display a directory of all checking accounts in the system.

2. Fidelity Savings and Loan has changed its name to First Fidelity Savings. Change both accounts at this bank to reflect this name change. (Use the CHANGE command.)

3. Close out Fran & Jerry's savings account at the Sic Semper Savings Bank.

4. Using the SHOW transaction, display a statement for the last month (September 1, 1978, through the end) in Jerry's checking account at the Sic Semper Savings Bank (account number 2-60-333).

   Close out the account (using the CLOSE task command).

5. Make the following changes to Fran's checking account at Prudential Bank & Trust:

   A deposit of one hundred dollars ($100), made on August 29th, was not recorded. Record the deposit.

   Check # 432, made out Aug. 20, should be in the amount of $25.40, rather than $24.50. Correct the entry (using the CORRECT transaction).

   Remove check number 427 ($85 for day care; 8/10/78). (Use the REMOVE transaction.)

Two checks were not recorded in the account; enter them now, using the WITHDRAW (or W) transaction. The checks are

421, August 3; $8 for gas
438, August 31; $15 for play tickets

List all checks for gas in August and September (between 8/1 and 9/30) using the LIST (or L) transaction.

6. Display a directory of all checking accounts.

7. Enter the following checks, deposits, and service charges in Jerry's checking account at Prudential Bank & Trust (account # 357-21001):

check no.	date	check issued to	check amount	deposit amount
863	9/5	newspaper	6.50	
864	9/12	ASL dues	17.00	
865	9/14	gas	5.00	
	9/15	service charge	.20	
866	9/15	watch repair	9.83	
	9/15	DEPOSIT		200.00

(Remember that you can use the CORRECT (C) transaction to correct any entries if you make a mistake.)

Display an account statement for the period covered by the entries you have just made. (Use the SHOW or S transaction.)

8. Open a savings account at Lake City Bank. The account number is 345-1101, interest is 6.5% compounded quarterly, and the account is to be in Fran's name. The account was opened on June 1, 1978. Record the following entries:

a deposit of $1,250 when the account was opened on June 1;

a deposit of $500.00 on August 15;

a withdrawal of $630.00 on September 20;

a deposit of $200 on September 25;

a withdrawal of fifty dollars ($50) on October 15.

9. Display a directory of all savings accounts.

10. Display a directory of all accounts at Sic Semper Savings.

Appendix H

This appendix summarizes data collected in the experiment. The first page summarizes subjects' responses to the questionnaire. The next page summarizes performance measures.

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	TQS
	4	4	5	5	5	5	5	3	5	5	41
	5	4	4	4	4	5	5	3	3	4	36
	4	4	5	5	4	5	5	3	4	5	39
	4	4	5	4	5	5	4	3	5	5	39
NO AIDS	5	4	5	5	4	5	5	4	5	4	41
	4	4	4	4	3	4	5	3	4	4	35
	4	5	4	4	4	4	4	3	3	4	35
	5	4	5	5	5	5	5	3	5	5	42
	4	4	5	5	5	5	5	2	2	5	37
	4	4	4	5	4	5	4	3	4	4	36
mean	4.3	4.1	4.6	4.6	4.3	4.8	4.7	3.0	4.0	4.5	38.1
s.d.	.483	.316	.516	.516	.675	.422	.483	.471	1.05	.527	2.64
	4	3	5	3	4	4	5	3	3	5	35
	4	4	5	5	4	5	5	2	5	5	39
	5	4	4	4	4	4	5	3	5	4	38
	5	4	5	5	4	5	5	3	3	4	38
HELP	4	5	5	5	5	5	5	4	5	5	43
	4	4	3	4	4	5	5	3	4	4	35
	4	2	4	4	3	4	5	2	2	3	29
	2	3	4	4	4	4	5	5	4	3	34
	4	4	4	4	5	3	4	5	5	5	40
	4	4	5	4	5	5	5	3	4	4	38
mean	4.0	3.7	4.4	4.2	4.2	4.4	4.9	3.3	4.0	4.2	36.9
s.d.	.816	.823	.699	.632	.632	.699	.316	1.06	1.05	.789	3.84
	4	4	4	4	4	5	5	4	4	5	38
	4	4	4	4	4	*	5	5	4	5	39
	4	5	5	5	5	*	5	5	4	5	43
	3	4	4	4	3	4	4	3	4	4	33
USER AIDS	4	4	5	5	4	*	5	5	5	5	42
	4	5	5	5	4	4	5	4	5	5	42
	4	4	4	4	5	5	5	5	5	4	40
	5	4	5	5	4	5	5	5	5	4	42
	3	4	5	4	3	*	5	5	4	4	37
	5	5	5	4	4	5	5	4	4	5	41
mean	4.0	4.3	4.6	4.4	4.0		4.9	4.5	4.4	4.6	39.7
s.d.	.667	.483	.516	.516	.667		.316	.707	.516	.516	3.06

* indicates manual was not used at all

	COST	INPT	CNCT	PRNT	TOT	UNQ	RPT
	1.36	1243	1795	858	5	5	0
	1.28	1155	2188	990	6	4	2
	1.45	1309	2274	990	5	4	1
	1.69	1299	2360	1122	8	4	4
NO	1.61	1188	3965	1056	2	2	0
AIDS	1.56	1144	3434	990	8	5	3
	1.72	1342	2816	1254	8	5	3
	1.34	1145	1822	858	2	2	0
	1.45	1199	2090	990	5	4	1
	1.54	1166	2622	1056	7	4	3
mean	1.50	1219	2537	1016	5.6	3.9	1.7
s.d.	.149	74.2	700	117	2.27	1.1	1.49
	1.67	1266	3388	1056	11	6	5
	1.41	1034	2175	858	4	4	0
	1.73	2090	2328	1452	8	6	2
	1.44	1232	1363	858	2	2	0
HELP	2.04	1453	2497	1320	6	6	0
MODE	1.65	1936	1724	1584	10	7	3
	1.58	1375	2130	990	9	5	4
	1.51	1265	1652	990	6	4	2
	2.01	1991	2530	1386	17	11	6
	1.84	2487	3028	1848	25	15	10
mean	1.69	1613	2281	1234	9.8	6.6	3.2
s.d.	.220	476	622	335	6.76	3.78	3.19
	1.50	1850	1840	1188	10	8	2
	1.85	1243	1656	1056	4	4	0
	1.09	1596	1101	1452	8	6	2
	1.90	1958	1934	1386	12	11	1
ON-LINE	1.79	1783	1350	1320	11	11	0
AIDS	1.56	1155	2235	990	6	6	0
	1.68	1463	1906	1518	7	6	1
	1.52	1310	1436	990	5	5	0
	1.36	1684	1901	1716	7	7	0
	.88	924	2111	858	1	1	0
mean	1.51	1496	1747	1247	7.1	6.5	.6
s.d.	.329	335	356	276	3.35	3.03	.843

COST: computing costs for entire session
INPT: approximate number of lines (commands & data) entered
CNCT: connect time (in seconds) to perform all tasks
PRNT: approximate number of lines printed by system
ERRORS: number of logical and syntax errors
  TOT: total number of errors
  UNQ: unique errors
  RPT: repeated errors

Appendix I

This appendix summarizes data collected in the experiment by grouping the NA (no aids) subjects with H (HELP mode) subjects who did not use the HELP mode.

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	TQS
	4	4	5	5	5	5	5	3	5	5	41
	5	4	4	4	4	5	5	3	3	4	36
	4	4	5	5	4	5	5	3	4	5	39
	4	4	5	4	5	5	4	3	5	5	39
NO	5	4	5	5	4	5	5	4	5	4	41
AIDS	4	4	4	4	3	4	5	3	4	4	35
	4	5	4	4	4	4	4	3	3	4	35
	5	4	5	5	5	5	5	3	5	5	42
	4	4	5	5	5	5	5	2	2	5	37
	4	4	4	5	4	5	4	3	4	4	36
	4	3	5	3	4	4	5	3	3	5	35
	4	4	5	5	4	5	5	2	5	5	39
DIDN'T	5	4	5	5	4	5	5	3	3	4	38
USE	4	5	5	5	5	5	5	4	5	5	43
HELP	4	4	3	4	4	5	5	3	4	4	35
	4	2	4	4	3	4	5	2	2	3	29
	2	3	4	4	4	4	5	5	4	3	34
mean	4.12	3.88	4.53	4.47	4.17	4.71	4.82	3.06	3.88	4.35	37.29
s.d.	.697	.697	.624	.624	.636	.470	.393	.748	1.05	.702	3.50
	4	4	4	4	4	5	5	4	4	5	38
	4	4	4	4	4	*	5	5	4	5	39
	4	5	5	5	5	*	5	5	4	5	43
	3	4	4	4	3	4	4	3	4	4	33
ON-LINE	4	4	5	5	4	*	5	5	5	5	42
AIDS	4	5	5	5	4	4	5	4	5	5	42
	4	4	4	4	5	5	5	5	5	4	40
	5	4	5	5	4	5	5	5	5	4	42
	3	4	5	4	3	*	5	5	4	4	37
	5	5	5	4	4	5	5	4	4	5	41
mean	4.0	4.3	4.6	4.4	4.0		4.9	4.5	4.4	4.6	39.7
s.d.	.667	.483	.516	.516	.667		.316	.707	.516	.516	3.06

Q1 through Q10: responses to questions 1 through 10
TQS: total of Q1 through Q10, except for Q6

* indicates manual was not used at all

	COST	INPT	CNCT	PRNT	ERRORS TOT	ERRORS UNQ	ERRORS RPT
	1.36	1243	1795	858	5	5	0
	1.28	1155	2188	990	6	4	2
	1.45	1309	2274	990	5	4	1
	1.69	1299	2360	1122	8	4	4
NO	1.61	1188	3965	1056	2	2	0
AIDS	1.56	1144	3434	990	8	5	3
	1.72	1342	2816	1254	8	5	3
	1.34	1145	1822	858	2	2	0
	1.45	1199	2090	990	5	4	1
	1.54	1166	2622	1056	7	4	3
	1.67	1266	3388	1056	11	6	5
	1.41	1034	2175	858	4	4	0
DIDN'T	1.44	1232	1363	858	2	2	0
USE	2.04	1453	2497	1320	6	6	0
HELP	1.65	1936	1724	1584	10	7	3
	1.58	1375	2130	990	9	5	4
	1.51	1265	1652	990	6	4	2
mean	1.55	1279	2370	1048	6.12	4.29	1.82
s.d.	.181	197	698	189	2.7	1.4	1.7
	1.50	1850	1840	1188	10	8	2
	1.85	1243	1656	1056	4	4	0
	1.09	1596	1101	1452	8	6	2
	1.90	1958	1934	1386	12	11	1
ON-LINE	1.79	1783	1350	1320	11	11	0
AIDS	1.56	1155	2235	990	6	6	0
	1.68	1463	1906	1518	7	6	1
	1.52	1310	1436	990	5	5	0
	1.36	1684	1901	1716	7	7	0
	.88	924	2111	858	1	1	0
mean	1.51	1496	1747	1247	7.1	6.5	.6
s.d.	.329	335	356	276	3.35	3.03	.843

COST: computing costs for entire session
INPT: approximate number of lines (commands & data) entered
CNCT: connect time (in seconds) to perform all tasks
PRNT: approximate number of lines printed by system
ERRORS: number of logical and syntax errors
- TOT: total number of errors
- UNQ: unique errors
- RPT: repeated errors

Appendix J

The following guidelines for message construction were developed in the course of implementing and using the on-line aids portion of the User Interface.

1) Empty messages should be carefully worded. It may not be entirely correct to tell a user that "no error has occurred" when the EXPLAIN ERROR aid is empty. Perhaps the empty condition represents an oversight on the part of the programmer who omitted a call to UISET, or the inability of a user to distinguish between prompts and error messages. These possibilities can, of course, be described in successive messages of the empty script. which can also refer the user to other aids.
2) Whenever possible, a message should end with an indication of what the user may do next. If a question was asked, the question should be repeated. Messages should indicate when additional messages in a script are available. For example, all but the last example in a series might end with "ENTER *eg FOR ADDITIONAL EXAMPLES."
3) User aids should not be used as an opportunity to make messages as short as possible. Even when an initial message must be brief, it can contain useful information. Messages should be specific (e.g., "FILE NAME MORE THAN 14 CHARACTERS" rather than "FILE NAME TOO LONG"); they

should indicate when input is ignored (e.g., "INVALID FIELD NAME; COMMAND IGNORED" instead of "INVALID FIELD NAME"), and they should indicate the form of the expected response (e.g., "DELETE? (Y or N)" rather than "DELETE?").

4) The user aid routines are designed to provide several different aids simultaneously, rather than a single HELP command. This design does not preclude having a command that explains the function and use of all the other user aids. Nor does it preclude the ability to associate a script with more than one aid. There may be situations, for example, when a request for EXPLAIN ERROR, EXPLAIN QUESTION, and EXPLAIN FORMAT can be satisfied with the same information.

5) Messages should be polite. They should not be anthropomorphized (e.g., "MY MEMORY BANKS ARE OVERLOADED"). Whenever possible, error messages should reflect system limitations rather than the user's inadequacy (e.g., "FILE XYZ CANNOT BE FOUND" instead of "NON-EXISTENT FILE: XYZ").

6) When several aids are provided, there should be a general-purpose aid that describes all other aids; the sign-on message should identify this aid. For example:

```
SYSTEM X VERSION Y
For available user aids, enter HELP.
```