Interface Design and Multivariate Analysis of UNIX Command Use

STEPHEN JOSÉ HANSON, ROBERT E. KRAUT, AND JAMES M. FARBER
Bell Laboratories

To understand how people interact with powerful computer systems, we analyzed, using several multivariate statistical analyses, the commands people use and the errors they made when performing office work with the UNIX operating system. The frequency of use across commands was very uneven. Users' most frequent commands were those that performed editing-like functions on text and other objects (e.g., UNIX directories), those that returned orienting information to users, and those that helped to control and sequence other commands. People made mistakes frequently, and made them most, when they needed information about the command and file context in which they were working, and when they had to plan long sequences of commands without feedback. From these analyses we make several recommendations for a human-computer interface.

Categories and Subject Descriptors: D.4.9 **[Operating Systems]:** Systems Programs and Utilities—*command and control languages*; H.1.2. **[Models and Principles]:** User/Machine Systems—*human information processing*; H.4.1. **[Information Systems Applications]:** Office Automation—*human factors, human computer interactions, man-machine interface*

General Terms: Experimentation, Human Factors, Measurement

Additional Key Words and Phrases: Human-computer interaction, command languages

1. INTRODUCTION

The patterns of behavior that people develop to work successfully with complex information systems are likely themselves to be complex. The standard methodology in the study of human-computer interaction is the traditional controlled experiment. Although this approach is appropriate and useful in theory-guided research, where the researcher has already identified the relevant variables and wants to know their causal relationship, it is less appropriate when the researcher needs to identify new variables or complex unknown relationships between new variables. Nor does it deal efficiently with highly multivariate phenomena such as human-computer interaction.

Portions of this paper appeared in the *Proceedings of the CHI'83 Conference on Human Factors in Computing Systems*, Dec. 1983.
The first two authors contributed equally to this research, and the order of authorship was determined randomly.
Authors' addresses: S. J. Hanson, 3D-468; R. E. Kraut, 3D-551 (of the Bell Operating Co.) CSO, Murray Hill, NJ 07974; J. M. Farber, 3J-319, AT&T Information Systems Laboratories, Lincroft, NJ 07738.
Correspondence should be sent to either S. J. Hanson or R. E. Kraut.

Where neither theory nor time will tolerate the isolation of a few controlling variables, assessing people's natural use of a computer system may be highly informative. Recent studies of document preparation [4, 7, 12, 19, 21], command-language use [3], and programming [17] are representative of this approach. Generally, observational data of human-computer interaction can allow the testing of simple hypothesis and intuitions, the discovery of computer features which cause problems for users, and guidelines for interface design.

Simply sampling the rich data that flow from human-computer interactions is not sufficient for their characterization. Also critical are the appropriate data reduction methods that extract patterns and represent the data in a simpler form [5, 10, 11]. Although such data have been collected [3, 21], these attempts have not included a complete analyses of the transition matrices which can result from the human-computer interaction. Together, observational methodologies and data reduction techniques can provide the researcher and designer with powerful tools for analyzing the human-computer interaction.

In applying this approach to the study of human-computer interaction, we focused our research on the UNIX operating system[1] We thus extend past research based on observational methodologies to the UNIX system. The UNIX operating system offers a widely used, powerful, and potentially complex computing environment to a wide range of users. From the office worker's point of view, the UNIX operating system has three components: (1) a hierarchical file system, in which files and directories can be nested arbitrarily deep in other directories; (2) a large number of elegant, generally useful tools to access, modify, and view text and other data; and (3) a command interpreter and programming language, called the *shell.* The shell allows users to tie together these resources by redirecting the output of commands to the screen, to files, or to other commands, and by forming procedures based on commands and files (called *shell scripts*).

2. DATA COLLECTION

We collected command usage data from the active users of a DEC VAX 11/780 computer running the UNIX operating system (Release 4.0). They represented a wide range of skill levels, from novice users to those with several years experience. The sample, however, was dominated by expert users who log-on daily and remain connected for several hours at a session. They were primarily office workers and professionals, including clerical workers, technical workers, programmers, and managers, whose main task was document preparation.

In order to collect a large number of commands unobtrusively, we used the UNIX system's accounting package, which automatically records data from each user about each process, in temporal order, executed during a log-on session. A *process* is the execution of a compiled program. It roughly corresponds to a command, with the exceptions noted below. These data were edited to eliminate processes that users do not typically execute directly (e.g., processes generated as part of system log-on or as part of intercomputer communication). Altogether we collected over 24,000 processes from 170 active users.

Process accounting records give an incomplete picture of user activity, recording only some commands, attributing to users processes that were invoked

[1] UNIX is a trademark of Bell Laboratories.

indirectly, and failing to record errors. To overcome these limitations, we supplemented the accounting data with command-line data from a smaller group of users. We recorded the commands that users issued from the keyboard and any error messages with which the system responded. Unlike the process accounting data, which were collected automatically, our procedure for collecting the command-line data had to be invoked by a user upon logging on. As a result, users were aware that their command use was being monitored. We collected about 11,000 commands in approximately 2 weeks from 16 members of a Bell Laboratory department, including data from social scientists, research assistants, and managers. They used the computer primarily for document preparation and for electronic communication.

In most of our analyses, the two data sets were analyzed together. Where they could not be merged, however (e.g., in the error analysis), they will be referred to respectively as the *process-data* and the *command-line data.*

3. RESEARCH QUESTIONS

As argued in the Introduction, it is important to plan the data reduction of human-computer interactions around specific questions or issues. We organized our data analysis to focus on three general areas of command usage:

3.1 Core Commands

What are the most important commands that users execute? Are some commands ubiquitous and essential for using other commands or for performing other actions? Answering these questions leads, in turn, to two additional concerns. First, we need to know the size and membership of the "effective" command set. This set contains the commands that people use most often and that occupy them longest. To identify these commands, we used a simple frequency analysis.

Second, we also need to characterize the use of commands in a more general way than that allowed by simply counting the frequency of particular commands. That is, we need to classify the types of commands present in the "effective" command set on the basis of the particular ways a command is used with other commands. This kind of analysis can determine how *modular* a command is and what functions it performs for the user. We classified commands using both our knowledge of the UNIX system and multivariate grouping techniques.

3.2 Usage Patterns

To understand how people use a computer system for office work, we also need to examine the sequential structure of commands, that is, how they lead to or follow from each other. This will give us information about the type and size of users' response units, the flow of commands, and the functions of command sequences, as they are used together. In addition, we can identify those commands that have many links with other commands, and through which the other commands are linked. To collect this information, we assessed the sequential dependencies among commands.

3.3 Errors

What causes users to make errors and how do users remedy them? We can attempt to answer these questions by examining the types of commands that lead to errors and their persistence. Since error information was not available in the process data, these analyses are based entirely on the command-line-data.

4. RESULTS AND DISCUSSION

This section summarizes the major results of our analyses and presents some observations on the use of complex computer systems for office work. More general implications of the results for interface design will be discussed in the following section.

4.1 Command Usage

4.1.1 *Distribution of Command Use.* Users of the UNIX operating system have a large number of commands available to them, and yet they used only a small proportion of these commands with any frequency. For example, although users had well over 400 commands available to them, in the process data, 10 percent of the commands accounted for almost 90 percent of the command usage. Similarly, in the command-line data the 20 most frequent commands, listed in Table I, accounted for about 70 percent of the usage. These statistics underestimate the concentration of command usage by ignoring sequences of commands that are sequentially dependent (see below) and treated as a unit by users.

The distribution of command use can provide more information about the way commands are organized and can set constraints on the relationships among commands. For example, if the command distribution were a step function, with one high frequency cluster and one low one, this distribution would imply that users have imposed a simple organization on the commands: Use the useful commands and ignore the rest. A smoothly decreasing distribution, however, indicates a more complex organization of command use.

Figure 1 shows the distribution of the most frequently used commands from the process data. These data are represented as a Zipf curve, with the frequency of occurrence of each command plotted against the rank of that frequency. In these log–log coordinates, the data show a smooth, regular decline in frequency, which is linear over most of the range.

These kinds of curves characterize both written [22] and spoken language [18]. Researchers usually argue that such curves are based on a hierarchically organized vocabulary of response units (i.e., commands) [15]. The smooth curve in Figure 1 is consistent with a model that constrains command retrieval to be interdependent.[2]

There are many reasons why most of the commands are used so infrequently: redundance among commands, difficulty of use, or the infrequent need for them. Whatever the reasons, the uneven distribution of command use suggests that computer systems should find ways to increase the prominence and ease of access to frequently used commands. The traditional command organization, in which a large number of unused commands is as prominent as a smaller number of

[2] These issues are controversial, although few would argue with the difference in organizational complexity between a step function and a smoothly decreasing function. Further, different functions may imply different retrieval processes. Hill [13], for example, has shown that different Zipf-like curves are uniquely associated with different underlying probability distributions of response units.

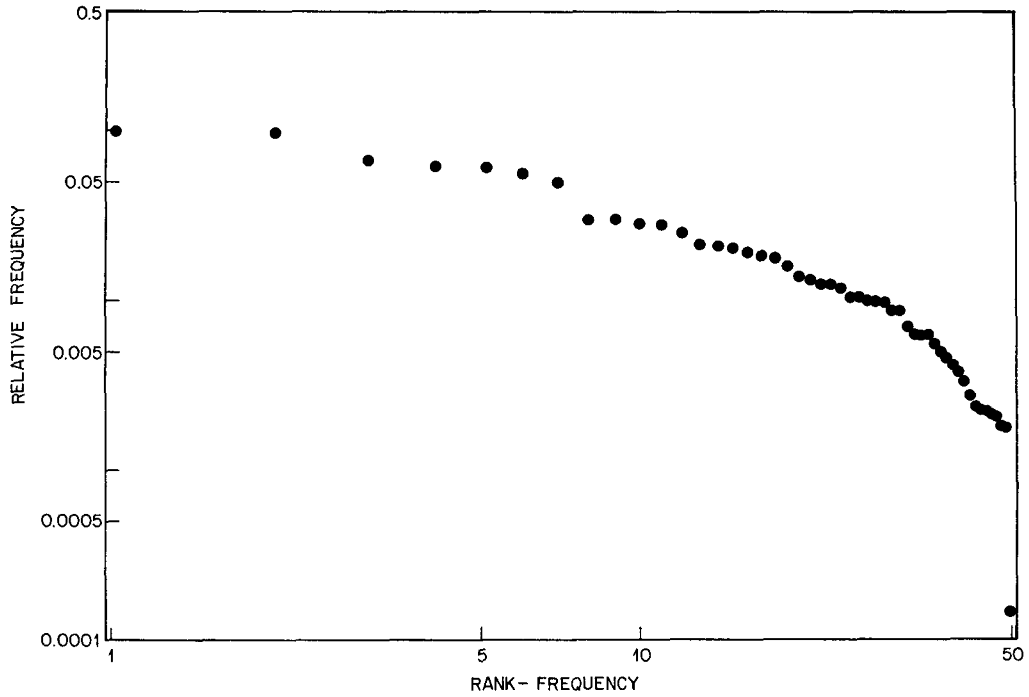

Fig. 1. A "Zipf curve" for the process command usage data.

frequently used ones, is likely to intimidate new users, hinder the learning of the command set, make identification and location of important commands difficult, and lead to erroneous command entry.

4.1.2 *Frequently Used Commands.* Table I shows the most frequently used commands in the command-line-data, along with a description of each command. The process data list is similar, but excludes commands that do not initiate a UNIX process (e.g., cd or |) and includes commands embedded frequently in shell scripts (e.g., echo and sed).

In Table I, we classify the commands according to the five major functions they serve, based on our knowledge of the UNIX operating system. This classification is also supported by the multivariate analyses below.

(1) *Generic editing commands* that shape text and other objects are the most frequent. We conceive of editing broadly to include both text editors that manipulate text through the actions of editing primitives such as delete, copy, move, insert, search for, and move cursor and stand-alone UNIX programs that parallel the editing primitives and perform equivalent actions on nontext objects, such as files and directories. For example, rm, cp, and mv deletes, copies, and moves a file, respectively; > redirects output into a file; grep searches for strings; and cd moves a user through the hierarchical file system.

These generic editing commands accounted for 36 percent of command usage in the command-line-data. This frequency, however, underestimates their impor-

Table I. The 20 Most Frequent Unix Commands From Command-Line Data

Command	Function	Proportion of All Commands ($N = 9934$)	Functional Group
cd	Change to a new directory	0.123	editing
is	List contents of directory	0.100	status
ca	Show contents of file	0.096	status
|	Pipe output of command to input of another	0.062	process
vi	Screen editor	0.059	editing
ed	Line editor	0.056	editing
rm	Delete file	0.038	editing
;	Execute commands sequentially	0.027	process
>	Redirect output of command to file	0.025	process
Mail	Send or read mail	0.020	communication
nroff	Format text file	0.015	task
mail	Send or read mail	0.014	communication
mv	Move or rename file	0.012	editing
grep	Find pattern in file	0.012	editing
col	Filter control characters from text	0.009	task
echo	Write a string to the screen	0.009	
&	Execute a command in the background	0.009	process
tail	Show end of a file	0.007	status
pwd	Identify current directory	0.007	status
awk	Pattern scanning and processing language	0.007	task

tance, since one call to a text editor may involve the user for several hours and produce many editing primitive commands.

(2) *Orienting commands* account for 21 percent of the command-line data. Orienting commands inform users about the environment in which they are working, the status of objects and processes in that environment and the consequences of their (own) actions. We include commands that tell users where they are in the file system (e.g., pwd), what files they have to work with (e.g., ls and cat), and what processes they are currently running (e.g., ps).

(3) *Process management commands* account for 10 percent of the command-line-data. These commands are used to integrate individual commands into more complex units (|), to sequence operations (;), and to provide independent parallel operations (&). The "break" command, which we could not record, terminates current processes. It is frequently used and adds to the importance of the process management category.

(4) *Social commands* allow people to exchange information with others through the various mail and news programs. They accounted for 3 percent of command usage in terms of frequency and considerably more in terms of duration.

(5) *Task specific commands* are required for fulfilling specific job assignments, and are the next most frequently used. Users in our primary samples worked in

document preparation environments; consequently, they frequently used document preparation tools. Programmers, analysts, statisticians, and other specialized workers would use similarly specialized tools. Document preparation commands, excluding editors, accounted for 3 percent of command use in terms of frequency, but substantially more in terms of duration.

4.2 Command Groupings

In this section we examine some additional ways of organizing commands to determine which are the most important to users. In the preceding section we addressed the question of importance by looking at the frequency of use of particular commands and of commands categorized on conceptual grounds. In this section we categorize commands in terms of the contexts in which they appear. Two commands can be similar because they are both modular (i.e., appear in many different contexts), or because both are used with the same other commands. We address both types of similarity.

Our analyses are based primarily on the process data, with supplemental information taken from the command-line-data. We constructed a transition matrix, collapsed over subjects, that represents the frequency with which one command is followed by every other command [1]. Starting with this transition matrix, we constructed correlational and distance measures of command similarity, based on the extent to which commands appear in the same context as others. Thus, if two commands are both preceded by and followed by the same other commands with similar probability, they are similar to each other by these measures.

We then used graphical, multivariate techniques [5], hierarchical cluster analysis [6], and principle components analysis to group commands on the basis of their similarity. Although these analyses produced similar results, our discussion of the graphical and cluster analysis will focus on command modularity, and our discussion of the principal components analysis will focus on functional command groupings.

4.2.1 *Command Modularity.* We can first ask about the similarity among commands in terms of their modularity. A command is modular to the extent that it precedes and follows a wide variety of other commands and is used in a variety of contexts. On the other hand, a command that precedes or follows only one or two other commands is part of a fixed command chain, restricted to a single task domain, and is not modular.

Figure 2 graphically compares three command transition profiles [2]. Each profile (or "star") represents the transitions between a target command and all others. To form a profile, each command was arbitrarily assigned a radial direction around a target command. The length of the rays in a profile indicates the frequency with which other commands precede the target or follow the target command. The length of each ray has been normalized to a proportion of the total frequency of the target command.

Visual analysis of the stars, as well as a hierarchical cluster analysis on the transitions, identified three modal profiles. Some commands, like ls (Figure 2(a)) and cat (Figure 2(b)) are star-shaped, showing that these commands are used frequently, before and after many other commands. Because of their ubiquity,

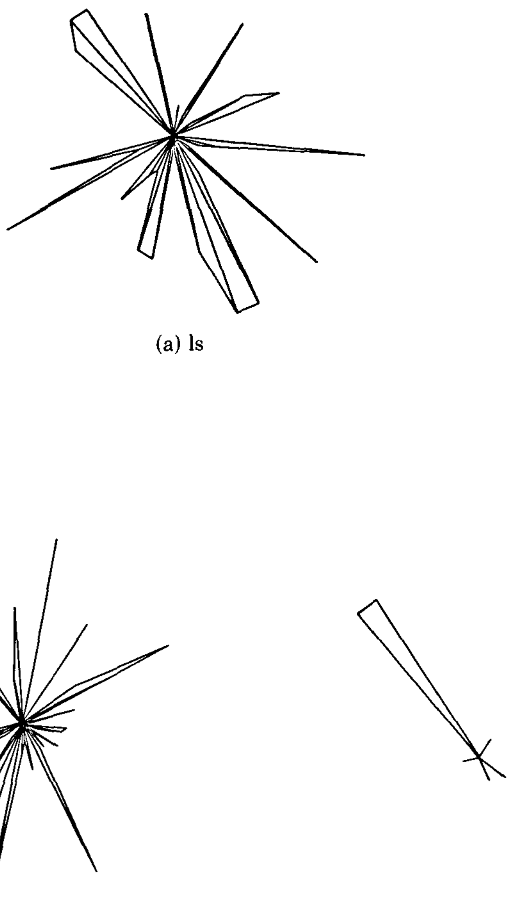

Fig. 2. Command profiles for the commands (a) ls, (b) cat, and (c) cp.

these commands can be termed *core* or *central* commands, regardless of their absolute frequency. They seem to provide support for other commands and are the most modular commands in the command space.

At the other extreme, some command profiles have only one or two spikes; these are commands that are used in the context of only one or two other commands in a highly stereotyped way. For example, cp, which copies a file, is generally preceded by chmod, which changes the permissions associated with a file, and also by itself (Figure 2(c)). These commands show little modularity; sequences of them tend to be treated by users as s single unit.

Finally, other commands are intermediate, used in stereotypic ways with a few other commands (e.g., mv, nrott).

4.2.2 *Functional Groupings.* To examine command groupings further, we applied a principal component analysis to the transition matrix. This method made stronger assumptions about the usage frequencies: First, that the variation in the

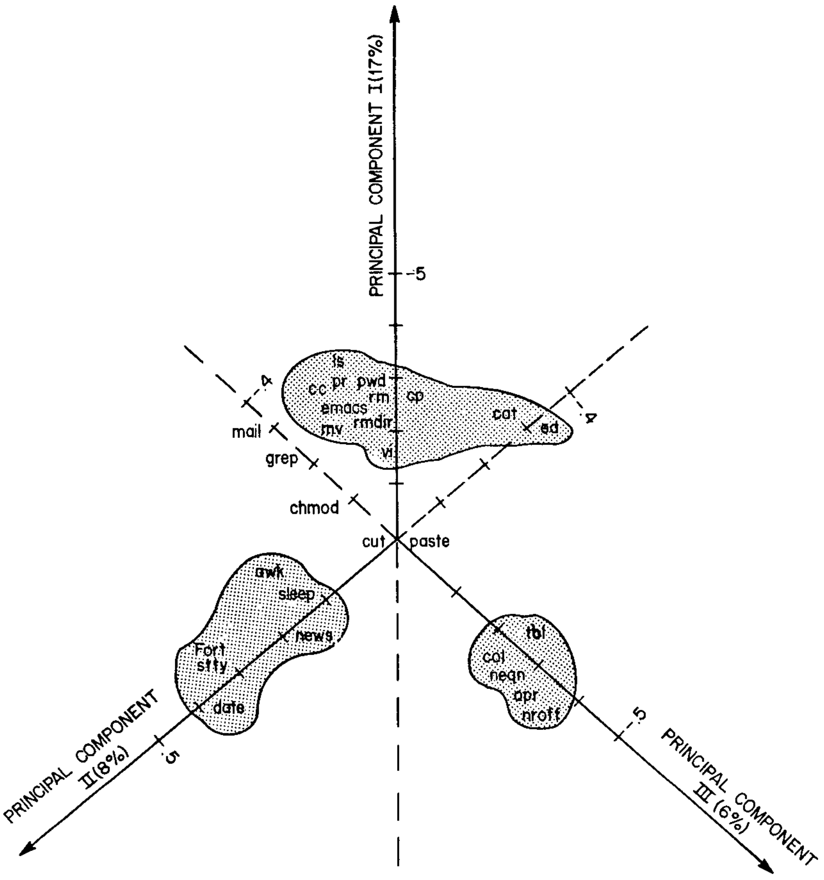

Fig. 3. First three principal components derived from a principal component analysis of command usage transition matrix.

transitions across other commands can be reduced to smaller sets of components, based on the linear combination of the transitions. And, second, that the transitions that comprise the components are normally distributed. The principal components analysis, based on the intercorrelations of the transition matrix frequencies, was used to extract, in sequence, each component that accounted for the greatest data variance.

Figure 3 shows the first three principal components and the commands that loaded on each component. The groupings derived from the principal component analysis are similar to those from the previous cluster analysis. For example, the first principal component consists of the modular, "star-shaped" commands. They are primarily the orienting and generic editing commands. The second group is task specific, containing commands for formatting and printing documents. And the third group contains commands associated with logging onto the computer system.

The total amount of variance that these three components account for is moderate (31 percent of the data variance) and is most likely due to violation of the strong linearity assumption of the principal components analysis.

4.2.3 *Usage Patterns.* Analyzing the transition matrix among commands as a contingency matrix [1, 14] allows us to determine what commands precede and follow what other commands at greater than chance levels, taking into account their absolute frequencies. A sequential dependency analysis reveals the stable patterns of use among commands.

Figure 4 shows the sequential structure of the UNIX command space. This representation provides information about the relative frequency of the commands, about the order in which they are used, and about their similarity. The sizes of the circles represent the probability of the corresponding commands.

The directional arrows show the temporal ordering of commands that significantly precede or follow each other. The stronger relationships are indicated by solid arrows ($p < 0.0001$) and weaker relationships are indicated by broken arrows ($p < 0.005$).

This representation confirms our descriptions of some commands as core and illustrates the command groupings we established earlier. Commands providing orienting information, implementing generic editing, and providing process management are central to this command space. For example, the frequency with which the ls command—which lists the contents of a directory—is used, its central position, and its 12 links with other commands suggest that people need directory information when issuing many other commands. For example, when users navigate their file systems (cd) or make changes to it (e.g., mv or cp), they list their files to confirm where they are and what they have done. The cat command, which shows the contents of files, is similarly frequent and highly connected (7 links).

The pipe command is also frequently used and has links to many other commands (16 links). It is the explicit mechanism in the UNIX operating system that is used to connect otherwise separate utility programs.

Notice that some links form closed paths for many of the commands in Figure 4. For example, ls links to cd, which in turn links to pwd, which leads back to ls. The number of closed paths (or circular links) can be used as a measure of the effect a command has on determining other command interactions. This measure is significantly high for ls (6 circular links) and cat (4 circular links), indicating that the orienting commands support and are central to the use of other commands.

Editing commands that move the cursor through the file system (cd), delete files and directories (rm and rmdir), copy files (cp), search for text strings (grep and egrep), and the text-editors themselves (ed and vi), are also central to the space, highly connected and have many closed paths.

Overall, the commands cluster in additional, more task-oriented, groups around the central orienting, editing, and process-management commands. These include a cluster to format and print text (nroff, neqn, tbl, pr, opr, and col), a cluster to check the spelling of text (deroff, spellpro, sed), and a cluster of social commands (Mail, mail, news).

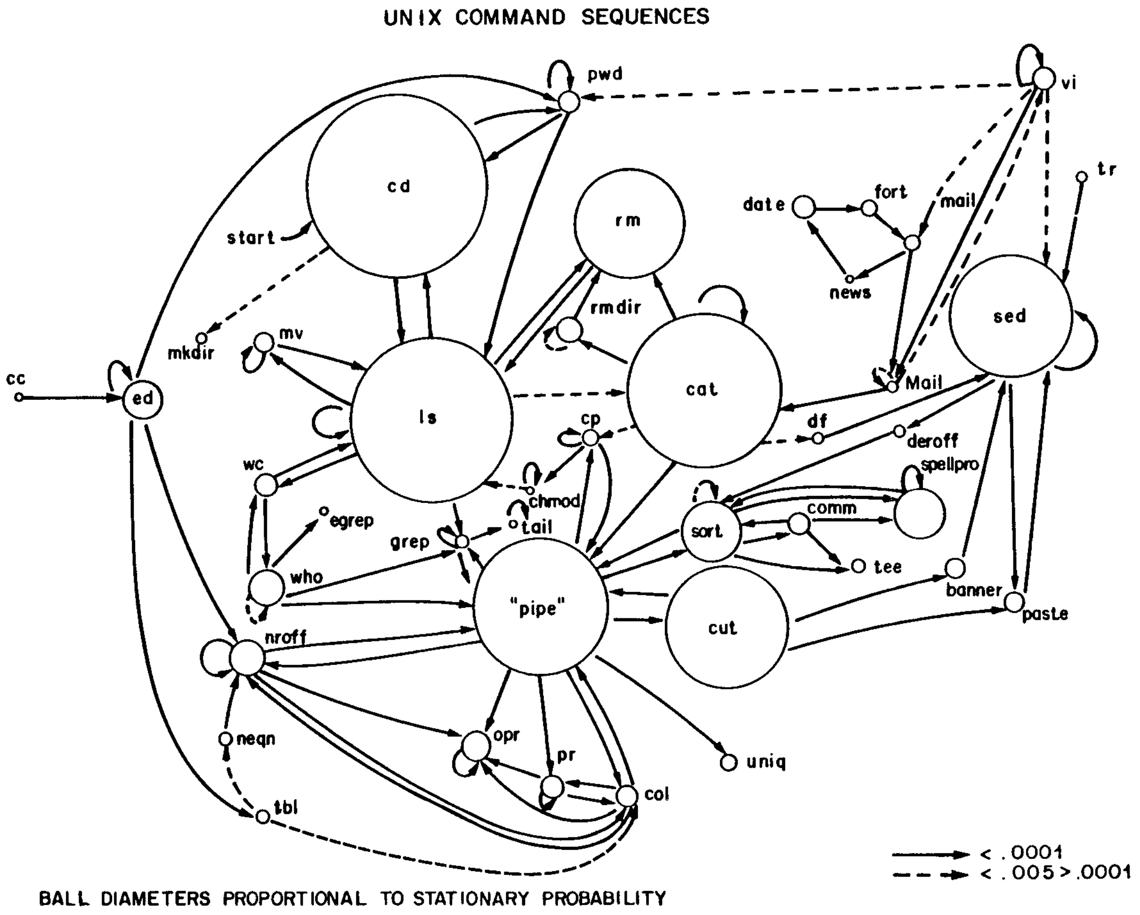

Fig. 4. Sequential structure of UNIX command usage. Each ball represents one of the 50 most frequently used commands in the UNIX command language. The size of the balls indicates the usage frequency of each command, and the arrows indicate the significant sequential dependencies, darker arrows being the more probable.

Table II. Selected Error Rates

Command	Frequency	Errors	Error Rate	Reason for Error
at	15	8	0.53	nonmodal
find	11	5	0.45	nonmodal
cc	22	9	0.40	planning
rmdir	22	8	0.36	status
awk	104	34	0.32	planning
nopr	16	5	0.31	?
cp	100	30	0.30	status
while	17	5	0.29	planning
for	51	14	0.27	planning
write	29	6	0.20	?
mv	159	30	0.18	status

4.3 Errors

In all computer systems, users make errors in trying to perform actions that the computer system does not recognize and that generate error messages. For example, when users try to print a nonexistent file, they clearly are making an error. In the command-line data, users made errors in 10 percent of the commands they issued. Error rates were very uneven across commands, ranging from less then 3 percent to over 50 percent. These figures underestimate the actual number of errors users made, since any system response that does not match the user's intention is an error. For example, users who print an unintended file have still made an error, although it is one that is unrecognized by the system. Table II shows the 11 commands from the command-line data that had the highest error rates, significantly above the mean ($p < 0.01$).

In analyzing why users make errors, we identified three classes of commands that produce errors at rates significantly greater than the mean error rate. The first are programming commands that require large amounts of planning before users get feedback about success. Planning without feedback is inherently difficult. More conversational programming languages, with feedback about intermediate actions, might decrease the overall error rate and the persistence of the errors.

The need for feedback is also true of programming constructs such as the UNIX *pipe*. The *pipe* is a mechanism to pass the output of one command to the input of another, without creating intermediate files. One of its limitations is its batch nature; a pipeline must be defined all at once. People use the pipe command frequently—it was the fourth most frequent command in the command-line data. They also, however, subvert it in order to get intermediate feedback about success. For example, instead of using the pipe, people frequently direct the output of a command to a temporary file and then use the temporary file as the input to the next command. Indeed, in more than half of the cases in which people redirect output to a file, the succeeding command uses that file as its input.

Users also made errors on commands that rely on their knowledge of status and orienting information, such as whether a file exists in the current directory, whether it is alterable, whether a name refers to a file or a directory, or whether a directory is populated or empty. Users often went to the expense of getting this status information. For example, as the sequential analysis shows, after changing

Table III. Distribution of Error Runs

Length	Frequency	Predicted*
1	678	895
2	101	89
3	37	9
4	13	1
5	11	0.1
>6	3	0.01

* geometric distribution (chi-square = 232, df = 2 $p < 0.01$)

directories with the cd command, users frequently looked for file information next, using the ls command (37 percent of the time, $r = 0.34$; see also Figure 4). But they also often made errors rather than incur these costs.

Finally, users made many errors on commands that were inconsistent with the modal command syntax. The modal UNIX command syntax is illustrated by the sequence "cut -cl-10 table 1," which displays the first ten columns of a file called (table 1). Users type a command name (cut), followed by hyphen-prefixed options and arguments (-cl-10), followed by filename arguments (table 1). By default, the output of the command is written to the user's terminal. Contrast the find command, which locates files with specified attributes in the file system (e.g., "find-name table 1-print"). It requires its argument to be indicated by a name flag, uses full-word rather than single letter option-flags, and defaults to no printed output.

A measure of the quality of feedback users get after making an error can be constructed from the sequential probability of making errors (i.e., the number of error "runs"). One might expect that good feedback ought to reduce the probability of sequential errors below that of chance. That is, the probability of making a second error following an initial error would be at least independent of the first error. In Table III we show the distribution of error runs compared to the predicted distribution, assuming independent errors (geometric distribution with $p = 0.10$ and $N = 9943$). This analysis shows that for two or more errors in a row, the probability of successive errors is *greater* than chance. Users are more likely to make an error when they try to reexecute a failed command than when they initially tried to execute it. The feedback that users get following an error does not often help them.

5. IMPLICATIONS FOR INTERFACE DESIGN

These findings lead to a number of suggestions for the design of the user interface to the UNIX operating system and for office automation of a similar power and kind.

5.1 Provide Functional Command Organization

In computer systems with large command sets, users need ways to set the availability or visibility of commands, and ways to find and learn about less frequently used commands. Providing core commands, a permeable menu organization, and a command thesaurus will help to solve these problems.

5.1.1 *Core Commands.* We have seen the occurrence of core commands in the frequency analysis and in each multivariate analysis. Some commands—those used for generic editing, those that provide orienting information, and those for process management—are used frequently in virtually all command contexts. This suggests that computer systems should be organized with a set of core commands always available to the user.

5.1.2 *Menus.* Other commands tend to be used less frequently, often in stereotyped groupings. The strong evidence of clustering and sequential dependencies among commands suggests that it would be practical to organize the commands around task-related menus. Menus reduce the amount of information about the command space that users have to keep in mind and make command selection easier.

Commands that are likely to be used in one context may also be needed in others. The use of a menu organization should not prevent users from assessing commands foreign to the current menu. In effect, command menus should serve as "help" facilities, reminding users of the options likely to be useful at a particular point, without barring them from other commands.

Default menus could be based on data of the sort we have collected, but individual differences in command usages will require that they be customizable by the user.

5.1.3 *Command Thesaurus.* When people are confronted with a large command set, they need ways to recall or find infrequently used commands. A command thesaurus that accepts a user's query and returns semantically related commands could fill this need [9].

5.2 Provide Relevant Orienting Information and Feedback

Some of the most frequently used and central commands provide the user with basic feedback on current status. In particular, users need to know the state of the objects they are working with and the effects of the changes they are making. They call for this information frequently and, when they don't get it, make errors. Such information is so important that it should be provided by default.

Providing essential orienting information automatically on a separate channel (e.g., in a reserved line or window) would dramatically reduce the number of commands users issue and would probably reduce the number of errors they make. At a minimum, the current command and current directory (including files) should be readily available, without the need to interrupt a task to obtain that information.

We see that when people plan sequences of operations, they make errors and frequently break up sequences of commands to get intermediate feedback. Users should be provided with conversational services to provide feedback. A conversational service (such as a conversational programming language, a screen-oriented text editor, or a spreadsheet) is one that lets users know the effects of one command or action before the user has to perform another.

5.3 Increase the Ease of Error Recovery

The structure of the errors in the data suggests a number of ways in which the error rate could be significantly reduced. As mentioned above, providing context information and intermediate feedback could eliminate many of the most common errors. Commands that violate the implicit rules for commands also lead to large error rates. Making command syntax consistent is an important step in reducing the error rate.

We must also design intelligent error handling that helps users to correct errors rather than compound them. For example, when users issue a command that cannot be executed, the system should prompt them with a choice of valid alternatives. If the command requested does not match any of the currently legal commands, the system should present the user with a choice of typographically and semantically related alternatives. Whenever possible, users should be able to edit and reexecute an erroneous command line.

6. SUMMARY AND CONCLUSIONS

In the present paper we have shown how to examine, extract, and organize properties of command language use. We found that people used commands very unevenly—using orienting commands, generic editing commands, and process management commands most frequently. These commands might be termed *core-commands*, since they are modular, linked with many other commands, and seem necessary for the other commands to be used effectively. Other commands were used less frequently, in set sequences, to perform particular tasks.

We also identified reasons why users make errors and why the errors persist—difficulties in planning, in getting orienting information, in dealing with inconsistencies, and inadequate feedback.

From our analyses, we argue that people can use a powerful computer system best if we provide them with immediate feedback about the consequences of their actions, with multiple views of the context in which they are working, with appropriately sized command sequences, with menus to help organize their commands, and with ease of error recovery.

We can integrate much of what we have said here by proposing a generic editor model for users' interaction with a computer system [8]. In this model, users could operate on objects, including text, menus, directories, and databases, through a series of generic editing commands, getting direct, visual feedback about the success of their actions, as they would in a good screen editor. In our editor model, objects and the properties of objects are visible and directly manipulable with immediate effect, similar to the way a person can point to a character in text and see it disappear when issuing a "delete character" command. We also mean that a focal or current object is shown in context, in much the way that good screen editors provide information about the current file name and cursor position within the file, while also showing the local editing changes.

This generic editor model integrates the task which users most frequently perform (i.e., text and file-system editing) with their general interactions with the computer. The model is consistent with people's frequent use of commands that perform editing functions and their consistent request for orienting information and feedback. It is also consistent with other recently proposed models [16, 20] of the human-computer interaction which emphasize the direct control and manipulation of the user's environment.

REFERENCES

1. BARTLETT, M.S. The frequency goodness-of-fit test for probability chains. *Proc. Cambridge Philos. Soc. 47* (1951), 89–95.
2. BECKER, R.A., AND CHAMBERS, J.M. S, a language and system for data analysis. Bell Laboratories, 1981.
3. BOIES, S.J. User behavior on an interactive computer system. *IBM Sys. J. 1* (1974), 2–18.
4. CARD, S.K., MORAN, T.P., AND NEWELL, A. *The Psychology of Human-Computer Interaction.* Lawrence Erlbaum Assoc., Hillsdale, N.J., 1983.
5. EVERITT, B. *Graphical Techniques of Multivariate Data.* North Holland, New York, 1978.
6. EVERITT, B. *Cluster Analysis.* Heinemann Educational Books Ltd., London, 1974.
7. FOLLEY, L.H., AND WILLIGES, R.C. User models of text editing command languages. In *Proc. Conference on Human Factors in Computer Systems* (Gaithersburg, Md, Mar. 15–17, 1982), ACM, Washington, D.C., 326–331.
8. FRASER, C. W. A generalized text editor. *Commun. ACM 23,* 3 (Mar. 1980), 154–158.
9. FURNAS, G., LANDAUER, T. K., GOMEZ, L., AND DUMAIS, S. Statistical semantics. *Bell Syst. Tech. J. 62,* 6 (July-Aug. 1983), 1753–1807.
10. GNANADESIKAN, R. Graphical methods for informal inference in multivariate data analysis. In *Proc. 39th Session, Bulletin of the International Statistical Institute* (1973), 195–206.
11. GNANADESIKAN, R., AND WILK, M. B. Data analytic methods in multivariate statistical analysis. In *Multivariate Analysis,* vol II, P.R. Krishnaiah, Ed., Academic Press, New York, 1969.
12. HAMMER, J. M., AND ROUSE, W. B. The human as a constrained optimal editor. *IEEE Trans. Syst. Man Cybern. 12,* 6 (Jan. 1982), 777–784.
13. HILL, B. The rank-frequency of Zipf's law. *J. Am. Statistical Assoc. 69,* 348 (Jan. 1974), 1017–1026.
14. LEMMON, R. E., AND CHATFIELD, R. E. Organization of song in cardinals. *Animal Behav. 19* (Feb. 1971), 1–17.
15. MANDELBROT, B. B. On the theory of word frequencies and on related Markovian models of discourse. In *Structure of Language in its Mathematical Aspect. Proc. 12th Symposium in Applied Math.* R. Jokobson, Ed., American Mathematical Society, Providence, R.I., 1961, 190–219.
16. NAKATANI, L. H., AND ROHRLICH, J. A. Soft machines: a philosophy of user–computer interface design. In *Proc. CHI'83 Human Factors in Computing Systems* (Boston, Dec. 13–15), 1983, ACM, New York.
17. NAUR, P. Program development studies based on diaries. In *The Psychology of Computer Use.* T.R.G. Green, S.J. Payne, and G.C. VanderVeer, Eds., Academic Press, New York, 1983.
18. RIDELY, D. R. Zipf's law in transcribed speech. *Psychol. Res. 44* (1982), 97–103.
19. ROBERTS, T., AND MORAN, T. The evaluation of text editors: methodology and empirical results. *Commun. ACM 26,* 4 (April 1983), 265–283.
20. SHNEIDERMAN, B. Direct manipulation: a step beyond programming languages. *Computer* (Aug. 1983), 57–69.
21. WHITESIDE, J., ARCHER, N., WIXON, D., AND GOOD, M. How do people really use text editors? In *Proc. ACM-SIGOA Conference on Information Systems* (Philadelphia, Pa., June 1982), ACM, New York, 29–40.
22. ZIPF, G. K. *Human Behavior and the Principle of Least Effort.* Addison-Wesley, Cambridge, Mass., 1949.

Received July 1983; revised August 1983; accepted September 1983