Directing The User Interface: How People Use Command-Based Computer Systems

Saul Greenberg and Ian H. Witten

Human/Machine Systems Laboratory
Department of Computer Science
The University of Calgary
Calgary, Alberta
Canada T2N 1N4
telephone: (403) 220-7140
uucp: ...!ihnp4!alberta!calgary!greenberg

Abstract — Several striking and surprising characteristics of how people use interactive systems are abstracted from a large body of recorded usage data. In particular, we examine frequencies of invocation of commands and complete command lines (including modifiers and arguments), as well as vocabulary growth. Individual differences are of particular interest, and the results are analyzed by user and by identifying groups of like users. The study underlines the remarkable diversity that exists even within groups having apparently similar needs.

Keywords — Command-based systems; computer interfaces; design principles; human-computer interaction; human factors; man-machine systems.

1 Introduction

Flexible interfaces create an environment in which users can pursue goals not addressed specifically by any one application package. They accomplish this by providing a rich set of primitive and high level actions and objects. Actions are traditionally invoked by users typing simple commands, although some modern systems augment or replace this primitive dialogue style with menus, forms, natural language, graphics, and so on [Witt85]. Typically, such interfaces either provide uniform access to all system actions or group these actions in some pre-defined way.

But when people use these top-level interfaces, they exhibit characteristic patterns of activity that are ill-supported by contemporary designs. Command-based interfaces, for example, provide uniform access to all system actions, whereas the actual usage of these commands is far from uniform. Although menus explicitly reveal pre-grouped system actions, they may not reflect the user's actual task organization. Similarly, "integrated" products may stress product rather than task integration [Niel86].

This work investigates how people direct command-based systems. It is based upon an analysis of long-term records of user-computer interaction with the UNIX *csh* command interpreter — a fairly popular and sophisticated system within the genre [Ritc74,Kern81] — collected as described in the following section. The subsequent sections provide statistical details of how people direct such systems in terms of how individual commands are used; the dependencies between them; and how complete command lines — commands, modifiers and arguments — are invoked. We are particularly sensitive to the fact that pooled statistics may conceal important differences between individuals, and the results are analyzed by user and by identifying groups of like users, as well as by pooling data for the entire population. We presume that the trends observed are shared by most command-based interactions, and are not just artifacts of a particular implementation.

2 Data collection

Command-line data was collected continuously for four months from users of the Berkeley 4.2 UNIX *csh* command interpreter [Joy80]. The start of every login session was noted, and all commands and arguments passed to *csh* were recorded sequentially exactly as typed. Each command entry was annotated with the current working directory, history and alias usage, and system errors (if any). From the user's point of view, the monitoring facility was unobtrusive — the modified command interpreter was identical in all visible respects to the standard version.

Four target groups were identified, representing a total of 168 users with a wide cross-section of computer experience and needs. Salient features of each group are described below, while the sample size (the number of people observed per group) is indicated at the bottom of Table 1.

Novice Programmers. Conscripted from an introductory Pascal course, these have little or no previous exposure to programming, operating systems, or UNIX-like command-based interfaces. Subjects spend most of their computer time learning how to program and use the basic system facilities.

Experienced Programmers. Members were senior Computer Science undergraduates, expected to have a fair knowledge of programming languages and the UNIX environment. As well as coding, word processing, and employing more advanced UNIX facilities to fulfill course requirements, subjects also use the system for social and exploratory purposes.

Computer Scientists. This group, comprised of faculty, graduates and researchers from the Department of Computer Science, is very familiar with UNIX. Tasks performed are less predictable and more varied than other groups, spanning advanced program development, research investigations, social communication, maintaining databases, word-processing, satisfying personal requirements, and so on.

Non-programmers. Word-processing and document preparation is the dominant activity of this group, made up of office staff and members of the Faculty of Environmental Design. Little program development occurs — tasks are usually performed with existing application packages. Knowledge of UNIX is the minimum required to get the job done.

Considerable variation was present in the number of commands entered by individual subjects (*mean* = 1712, *std dev* = 1499). As highly-active users can dominate the study [Hans84], we are careful to check for such bias when processing the data.

3 Distribution of command usage

3.1 Frequency distributions of commands for large groups

Several investigators have examined the frequency of command usage by a user population [Hans84] [Krau83] [Peac82]. All studies report results approximated by a Zipf distribution [Zipf49] [Witt84], which has the property that:

- a relatively small number of items have a high usage frequency; and
- a very large number of items have a low usage frequency.

A looser characteristic of this kind of rank distribution is the well-known 80-20 rule — 20% of the items in question are used 80% of the time [Knut73,Peac82]. In measurements recorded from a UNIX site, [Hans84] report a similar result — 10% of the 400-500 commands available account for 90% of the usage. These models also holds for the frequency distribution of all help requests made for particular commands through the UNIX on-line manual [Gree84][1].

The current study supports these observations. Figure 1 illustrates the command frequency distribution for the four different user groups. The vertical axis shows the number of command invocations, normalized to one for the most frequent, while the horizontal axis shows the rank ordering of commands, with the most frequent first. The normalized Zipf distribution, calculated as $y = x^{-1}$ and illustrated by the smooth curve in the Figure, seems to provide a plausible model for the observed frequencies. For each of the four user groups, 10% of the commands used accounted for 84–91% of all usage (*cf* [Hans84]s' 10%/90%). This ratio seems independent of the actual number of unique commands selected or the size of the sample group.

3.2 Usage frequency of particular commands between groups

Even though frequency statistics of different groups are modelled by the Zipf distribution, we do not know whether commands retain their same rank order between different user groups. If they do, then a command used frequently by one group will have the same relative usage in another. As we shall see, this is not necessarily the case.

Table 1 shows the data from which Figure 1 is drawn. Each column shows the 20 most frequently used commands by each group (including data reported by [Hans84]) and also gives the total commands executed, the number of those which are unique, and the number of users sampled. The few common high frequency commands across the five user groups are mostly concerned with navigating, manipulating and finding information about the file store (such as *ls*, *rm* and *cd*). Comparison of other commands capture the differences between the groups. The emphasis on programming by both our novice and experienced subjects is reflected by the various compilers used (*pix* and *pi* for Pascal, *make* for "C", and *ada*). The non-programmers, on the other hand, seem concerned with word processing (as indicated by the relatively heavy use of *nroff* and *spell*). The type of editor also indicates group differences — *vi* and *ed* are chosen by [Hans84]s' group, while *emacs*, *e*, *umacs*, *fred*, and *ed* have varying degrees of use within the others.

Grouping all subjects into one category also illustrates the danger of using a population stereotype to approximate the activity in each group. As shown by column 1 of Table 1, which pools all subjects of this study into one large sample, some high-frequency commands are not used frequently (if at all) by all groups (eg *pix*, *umacs*).

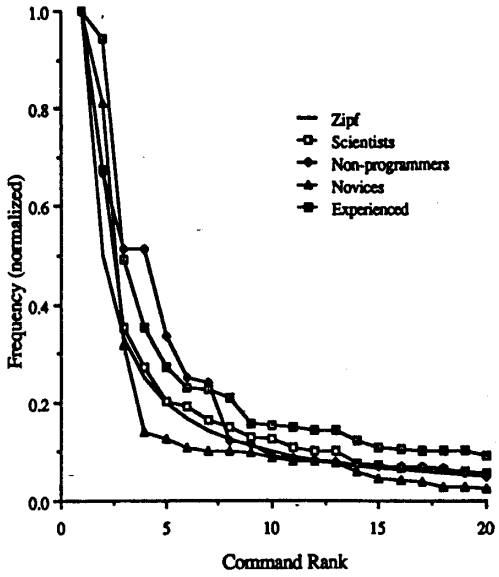

Figure 1. The normalized command frequency, compared with Zipf

Even though the Zipf form of the frequency distribution remains intact between different groups of a population, the rank order of commands is not, in general, maintained.

3.3 Frequency distributions and command overlap between individuals

We have not yet considered the extent to which the usage statistics of an individual resembles those of a group of like people. Does the Zipf distribution characterize each user's command interactions, or is it just an artifact of data grouping? Do individuals within a group invoke the same set of commands? One might expect the variation between users to be even greater than that between groups.

In the same study of the UNIX on-line manual noted previously, the frequency distribution of help requests was analysed between individuals [Gree84]. In general, users constrained themselves to relatively small subsets of the requests possible — a great many potential entries were never accessed by them. Moreover, when users' subsets were compared, the intersection between their elements was small. Additionally, the frequency of access of the common elements varied considerably across users. [Gree84] suggests that although individual help requests seem to follow the Zipf distribution, it is not possible to make any but the grossest generalization from a population perspective of how each individual user will access particular items within a system.

The same is true for command line interactions. While studying the nature of expertise in UNIX, [Drap84] counted the times a command was invoked by each user. First, out of a vocabulary of the 570 commands available to the population, only 394 were used at least once. Individuals knew the system to varying degrees — there was a fairly smooth distribution of vocabulary size up to the maximum of 236 commands known to one user. Characteristics of the overlap between individuals' vocabularies were similar to those of [Gree84]'s study. Generally, very few of each individual's commands were used by all the population, a few more shared to some degree by other users, and the rest used by him alone. [Drap84] concluded that vocabulary is a poor

[1] Every command in the Unix system has a corresponding manual entry, invoked by typing *man <command>*

Groups from the Current Study											Others	
All Subjects		Novice Programmers		Experienced Programmers		Computer Scientists		Non-Programmers			[Hans84]'s Group	
command	% used	command	% used	command	% used	command	% used	command	% used		command	% used
ls	13.33	pix	25.64	ls	12.76	ls	15.75	ls	18.53		cd	12.30
cd	8.83	umacs	20.89	cd	12.03	cd	10.62	emacs	12.35		ls	10.0
pix	6.69	ls	8.18	e	6.29	e	5.58	cd	9.56		cat	9.6
umacs	5.34	rm	3.55	fg	4.42	fg	4.32	nroff	9.55		|	6.2
e	4.47	u	3.19	more	3.49	rm	3.21	e	6.20		vi	5.9
rm	3.35	cat	2.79	make	2.93	mail	3.00	rm	4.66		ed	5.6
emacs	3.28	more	2.63	rm	2.93	emacs	2.58	ee	4.47		rm	3.8
fg	3.07	cd	2.61	emacs	2.66	lpq	2.36	lpq	2.25		;	2.7
more	2.51	script	2.49	l	2.02	more	2.06	ps	2.13		>	2.5
lpq	2.02	lpr	2.26	cat	1.96	ps	1.97	cp	1.66		Mail	2.0
mail	1.95	cp	2.09	lpr	1.91	f	1.70	ptroff	1.65		nroff	1.5
cat	1.89	lpq	2.08	ada	1.85	cat	1.62	more	1.59		mail	2.0
lpr	1.49	emacs	1.95	ex-vax	1.85	who	1.59	w	1.50		mv	1.2
cp	1.48	pi	1.54	cp	1.58	mv	1.20	mail	1.37		grep	1.2
ps	1.36	p	1.21	rwho	1.37	man	1.18	rr	1.31		col	0.9
who	1.14	fred	1.04	a.out	1.33	rlogin	1.05	tbl	1.27		echo	0.9
make	1.08	mail	1.03	mail	1.31	cp	1.02	spell	1.27		&	0.9
nroff	1.06	pdpas	0.72	lpq	1.31	fred	0.99	mv	1.20		tail	0.7
fred	0.95	logout	0.71	ps	1.30	lpr	0.91	ed	1.02		pwd	0.7
man	0.90	pdp60	0.67	who	1.16	page	0.90	apq	0.89		awk	0.7
commands executed	287736	commands executed	73288	commands executed	70234	commands executed	119557	commands executed	24657		commands[9] executed	9934
unique commands	1307	unique commands	264	unique commands	588	unique commands	851	unique commands	196		unique commands	400
sample size	168	sample size	55	sample size	36	sample size	52	sample size	25		sample size	16

Table 1. Command distributions for the top 20 commands of five different user groups

measure of expertise, and that each user is actually a specialist in a particular corner of the system.

[Drap84]'s record of command activity was estimated by noting the UNIX processes spawned during a user's interaction with the system, unlike the present study which collects the actual text typed. Although a reasonable correspondence is expected between the command entered and the process created, information about name aliases and commands built into the UNIX interpreter is lost, even though some are frequently used. So is the distinction between processes created indirectly through side effects or through a user's command scripts or programs. Although these problems were recognized[2], it was suggested that overall trends and relative vocabulary sizes are probably representative.

Our study corroborates the conclusions of [Drap84]. Each row of Table 2 shows the proportion of commands shared by the users comprising a particular group, while Table 3 lists some of the actual commands shared. For example, only 0.2% of the 1307 different commands used by all subjects were shared by more than 90% of them (these were basic file manipulation commands for listing, removing and copying files, as shown in column 1 of Table 3). More surprisingly, fully 92% of all commands were shared by fewer than 10% of the users, if at all. These differences are much stronger than those suggested by [Drap84]'s group (the last column), probably due to inaccuracies in his estimate of command use.

Tables 2 and 3 also reveal that categorizing like subjects into groups changes the figures less than one might expect. For example, even though individuals in the novice group used the system for solving the same programming assignments and were taught UNIX together, there was relatively little intersection of their vocabularies. Except for a handful of commands, users — even those with apparently similar task requirements and expertise — have surprisingly little vocabulary overlap.

3.4 Growth of the command vocabulary

In the previous discussion, a user's vocabulary was taken to be the set of commands he invoked over a fixed period of time. But how dynamic is the command vocabulary of a user? Do users learn new commands sporadically or uniformly over time? Are new commands acquired continually, or do users stop increasing this vocabulary after some period of time?

Figure 2 illustrates the growth rate of the command vocabulary of four typical users, one from each group. The vertical axis is vocabulary size, while the horizontal axis represents the number of command lines entered so far. At first, the vocabulary growth rate seems to be around 5% — each user shown here has a repertoire of 43 – 64 commands after 1000 full command lines were entered. But the growth rate drops quickly afterwards to 1% or less. The later part of the curve is probably a better reflection of vocabulary acquisition, for the first part does not necessarily reflect a learning curve. Since users already knew a command subset before monitoring began, we would expect unusually high initial activity as known commands are being noticed for the first time.

Although Figure 2 suggests that the selected subjects have a vocabulary growth rate which is proportional to the relative sophistication of the group, analysis of variance shows no statistically significant differences between the mean rate of each group. However, these rates were determined by counting the new commands acquired between 1000 and 2000 command entries, which meant excluding those subjects who did not have least 2000 entries.

Figure 2 also reveals how users acquire new commands. Although there are short term periods where vocabulary growth is relatively uniform, there are also long periods of quiescence followed by a flurry of activity. As might be expected, this was sometimes associated with new tasks. For example, the sharp increase in new activity for the Scientist subject after she had entered 6000 command lines all involved high-quality typesetting (Figure 2). However, there are other instances where no such task association is evident.

4 Relations in command sequences

4.1 Dependencies in command sequences

The previous discussion says nothing about possible relations and dependencies between commands. Through a multivari-

[2] See [Bann84] for a description of similar problems.

# of users sharing a command (%)	Proportional Number of Commands Shared (%)					
	All Subjects	Novice Programmers	Exper'd Programmers	Computer Scientists	Non Programmers	[Drapé4]'s Group
100–91	.2	2.7	2.2	.9	1.5	.5
90–81	.3	.8	.7	.8	0	2.0
80–71	.3	.4	1.0	.8	2.0	3.1
70–61	.4	.8	1.0	.6	.5	3.3
60–51	.5	1.5	2.2	1.9	4.6	3.1
50–41	.5	2.7	1.9	1.1	3.1	6.1
40–31	1.2	0	1.2	1.4	4.6	6.1
30–21	1.5	9.1	4.1	4.4	6.6	8.6
20–11	3.0	12.1	8.9	6.5	34.7	17.8
10–1	92.0	70.1	76.9	81.7	42.4	49.5
Total unique commands	1307	264	588	851	196	394

Table 2. Number of users per command

The 20 Most Shared Commands										
All Subjects		Novice Programmers		Experienced Programmers		Computer Scientists		Non-Programmers	
command	# of users	command	# of users	command	# of users	command	# of users	command	# of users
ls	168	lpr	55	cd	36	ls	52	ls	25
rm	164	ls	55	ls	36	rm	51	rm	24
cp	154	pix	55	more	36	cat	50	emacs	23
lpq	149	rm	55	lpq	35	cd	50	cd	19
lpr	144	script	55	man	35	mv	49	cp	19
cd	141	cp	53	cat	34	cp	48	nroff	18
cat	140	lpq	53	cp	34	mail	48	lpq	17
mail	131	umacs	47	lpr	34	man	48	ps	16
more	130	cat	46	mail	34	mkdir	46	lpr	14
man	124	more	42	mkdir	34	ftp	44	more	14
who	117	cd	36	rm	34	lpq	44	logout	13
mv	114	mail	36	ftp	33	ps	44	mail	13
emacs	112	limits	32	ps	32	pwd	44	man	13
mkdir	104	who	30	mv	31	who	44	hpq	12
ps	103	man	28	who	31	fg	42	mv	12
fg	95	pi	28	ruptime	30	e	41	spell	12
script	95	logout	26	fg	29	emacs	41	who	12
pwd	92	help	24	kill	28	lpr	41	kill	11
ftp	91	lquota	23	limits	28	rlogin	40	pwd	11
logout	88	emacs	23	rwho	28	kill	38	cat	10
sample size	168	sample size	55	sample size	36	sample size	52	sample size	25

Table 3. The 20 most shared commands per user group

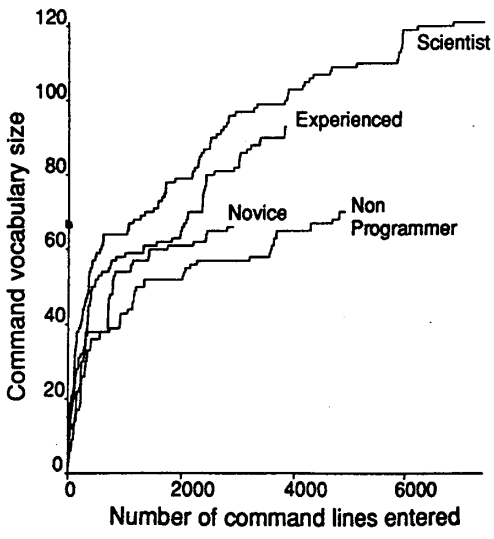

Figure 2. Command vocabulary size versus the number of command lines entered for four individuals

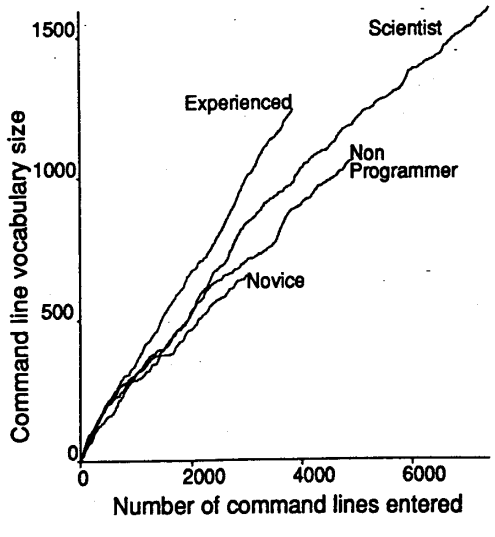

Figure 3. Command line vocabulary size versus the number of commands entered for four individuals

ate analysis of UNIX commands invoked by the site population, [Hans84] examined the interaction effects between commands. Their results show statistically significant relationships between certain command chains. One dimension of these relationships is *modularity*. Some commands are core commands — they are used frequently and are surrounded by many other commands (ie highly modular and independent). Others are not; they are surrounded by specific command sequences.

Commands are also related by functional clusters, such as: editing, process-management, orientation, social communication, and so on [Hans84], which may not be revealed by statistics. Consider a user who prints files in several ways: a short draft may go to the screen; a long listing to a lineprinter; and a final version to a laser printer. Although these non-sequential and possibly rarely invoked actions are related by function in the user's mind, it is unlikely that such a relationship would appear from a multivariate analysis of commands like [Hans84]s'. Additionally, it is a mistake to assume that all dependencies revealed by analysing a group of users will hold for an individual, since each person uses their own particular subset of commands (Section 3.3).

4.2 Grouping commands by user goals

Perhaps a more meaningful way of considering command dependencies is by relating them to particular tasks. [Bann83] analysed activities on a UNIX system by asking users to annotate periodically their command histories with their intentions. The data collected showed that it is possible to partition user actions and the objects they manipulate (such as files) into task sets related by the user's goals. When users are observed from this perspective, patterns appear which are not evident at the individual command level. For example, tasks are not invoked sequentially, but are interleaved due to the user switching, suspending and resuming his goals.

But [Bann83]s' study does not indicate how task sets differ between users. Perhaps a clue can be gleaned from the work of [Niel86], who investigated integrated software usage by professionals in a work environment. They tried to identify the main goals and subgoals of professionals and the methods used by them for satisfying their goals. Data was collected through questionnaires and interviews:

- Five high-level programs accounted for 42% of program use over the population.
- It was not possible to rank the programs accounting for the other 58% at the population level, as most were used by only a few professionals each.
- Integrated packages were not exploited fully. For example, users chose non-integrated modules if they were judged more effective in terms of goal achievement than the integrated version, ie users were "choosing a set of heterogeneous programs and integrating them in their own way" [Niel86].

[Niel86] concludes that integrated programs are not a panacea for communicating with general purpose computers, for "most current analyses have not yet developed categories of representation adequate for identifying the task requirements of integration" (p167). Even so, the high number of disparate programs used by professionals and the different ways they were "integrated in their own way" suggests that there are both subtle and overt differences between the task sets of users.

5 Command lines

As well as examining the commands users invoke, it is fruitful to look at complete command lines. We define a *command line* as the complete line (up to a terminating carriage return) entered by the user. This is a natural unit because commands are only interpreted by the system when the return key is typed. Command lines typically comprise an action (the command), an object (*eg* files, strings) and modifiers (options). In the following discussion, a sequential record of command lines entered by a user over time, ignoring boundaries between login sessions, is called a *history list*. Unless stated otherwise, the history list is a true record of every single line typed. The *distance* between two command lines is the difference between their positions on the list. The number of different entries in the history list is the *command line vocabulary*. Although white space is ignored, syntactically different but semantically identical lines are considered distinct[3].

5.1 Vocabulary and recurrences of command lines

Although it is known that only a small set of commands accounts for all user actions (Section 3), it is not known how often new command lines are formed and old ones recur. One might expect that they would recur infrequently, given the limitless possibilities and combinations of commands, modifiers and arguments.

Surprisingly, this is not the case. Although users extend their vocabulary of command lines continuously and uniformly over the duration of an interaction, the majority of lines entered are recurrences, where the mean recurrence rate for groups ranges between 68% and 80%, with Novice Programmers exhibiting the highest scores. Although an analysis of variance of the raw scores rejected the null hypothesis that the group scores are equal, it is reasonable to approximate the recurrence rate by the population mean of 74%. That is, about three out of every four lines entered by a user already exist on the history list. Conversely, an average of one out of every four appears for the first time.

The formation of new command lines is surprisingly linear and regular, as illustrated by Figure 3. Similar to Figure 2, and using the same typical users, the horizontal axis still represents the number of lines entered so far, but now the vertical axis indicates the size of the command line vocabulary. For example, the scientist subject has composed close to 1400 new command lines after 6000 lines were entered. The long periods of quiescence and the flurries of new activity seen in Figure 2 are notably absent from Figure 3.

5.2 Command line frequency as a function of distance

For any command line entered by a user, the probability that it has been entered previously is quite high. But what is the probability distribution of that recurrence over each previous input? Are recurrence distances, for example, spread uniformly across the distribution or skewed to the most recently entered items?

The recurrence distribution as a measure of distance was calculated for each user, and group means are plotted in Figure 4. The vertical axis represents the rate of command line recurrences, while the horizontal axis shows the position of the repeated line on the history list relative to the current one. Taking Novice Programmers, for example, there is an 11% probability that the current command line is a repeat of the previous entry (distance = 1), 28% for a distance of two, and so on. The most striking feature of the Figure is the extreme recency of the distribution.

The previous seven or so inputs contribute the vast majority of recurrences. It is not the last but the second to last command line that dominates the distribution. The first and third are roughly the same, while the fourth through seventh give small

[3] This aspect of our study is reported in greater detail in a companion paper, which develops principles of design for history mechanisms [Gree88].

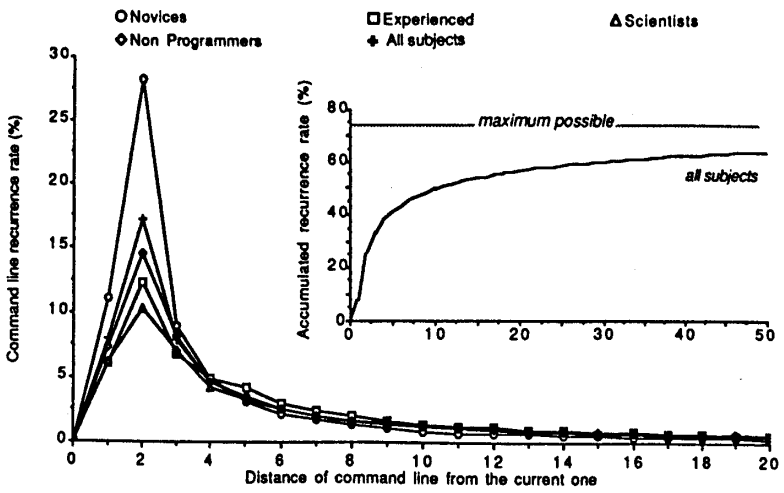

Figure 4. The recurrence distribution of command lines as a measure of distance

but significant contributions. Although probability values continually decrease after the second item, the rate of decrease and the low values make all distances beyond the previous ten items practically equivalent. This is illustrated further in the inset of Figure 4, which plots the same data for the pooled groups as a running sum of the probability over a wider range of distances. The most recently entered command lines on the history list are responsible for most of the accumulated probabilities. In comparison, all further contributions are slight (although their sum total is not). The horizontal line at the top represents a ceiling to the recurrence rate, as 26% of all command lines entered are first occurrences.

Figure 4 also shows that the differing recurrence rate between user groups can be attributed to the three previous command lines. Recurrence rates are practically identical elsewhere in the distribution. This difference is strongest on the second to last input, the probability ranging from a low of 10% for Scientists to a high of 28% for Novice Programmers.

6 Discussion

This paper has analyzed new data in several new areas on user interactions with command-based computer systems and has surveyed other related works. The major conclusions are:

1. The rank frequency distribution of command usage by groups of like and unlike users is approximated by a Zipf distribution.
2. With a few exceptions, the frequency of use of most commands differs between groups — rank order is not maintained.
3. There is little overlap between the command vocabulary of different users, even those with apparently similar task requirements and expertise.
4. Users acquire new commands slowly and irregularly.
5. Some commands cluster around or follow others in statistically significant ways.
6. User activity can be partitioned into task sets related by the user's goals.
7. There are differences between the task sets of users.
8. A substantial portion (~75%) of each user's previous command lines are repeated.
9. New command lines are composed regularly (~25% of all lines).
10. Users exhibit considerable recency when repeating command lines.

These conclusions have important implications for system design, and although space does not permit a comprehensive discussion, we briefly indicate three areas where they could have an immediate impact: intelligent tutoring systems; fine-tuning existing implementations; and design of personalized workspaces.

Intelligent tutoring systems. Models of command-sets used for a task can become part of an intelligent tutoring or coaching system. One approach for deciding what knowledge should be presented to the user employs an "expert" and a "student" model [Slee82]. For example, the *differential* model of [Burt81] bases its instructional presentation on the differences between a student's and an expert's behaviour. But according to the present study, it is not possible to decide what commands should be offered to the student, since experienced users of general purpose systems such as UNIX don't seem to share particular command sets. Consequently, the differential model is not appropriate for teaching people how to use general purpose computer systems.

Fine-tuning existing implementations. One way of evaluating and then incrementally fine-tuning an implementation is through case studies of users interacting with the system. A user's actions and mistakes are observed, and the original system design is modified accordingly. For example, the extremely heavy use by all users of the basic file manipulation commands (noted in Table 1 and Section 3.3) indicates that users require not only constant feedback on the contents of the current directory, but some simple tools for manipulating them. Feedback of the directory contents can be provided by keeping a permanent display of the current files on view, a simple task given a window-based environment. If screen real-estate is a concern, transient windows popped up by a mouse press may be used instead [Gree86]. These findings also support the inclusion of the more sophisticated file browsers that are found on many modern systems.

Personalized workspaces. A more innovative approach is to re-structure command sequences as *workspaces* which group related commands into a visible structure [Bann83] [Hans84] [Gree85]. But who should create these workspaces? In ME-NUNIX [Perl84], for example, it is difficult to reconfigure the workspaces supplied with the system. Yet the results of the previous sections indicate that this provides limited benefit. Personalizable workspaces seem much more promising. For example, the *workbench creation system* uses a direct manipulation

editor that allows rapid creation and modification of hierarchies of workspaces by the end-user [Gree85]. Using this mechanism, a simple default structured workspace can be appended to the top-level command system by the system manager. Group leaders can then modify this structure to reflect common local needs. With a set of sensible defaults in place, individuals can alter the workspace further to fit personal needs. For example, a novice using the system for document creation and management may have available a generic workspace that includes facilities such as an editor, various printers, spelling and style checkers, and so on. When this user becomes more adept, he may alter the structure to reflect his own needs. And since recently entered commands have a high probability of being repeated, a history mechanism could offer likely candidates for the workspace, eliminating the need for their recomposition [Gree88].

Acknowledgement

We give special thanks to the many volunteer subjects who let us observe their system use. This research is supported by the Natural Sciences and Engineering Research Council of Canada.

References

[Bann83] Bannon, L., Cypher, A., Greenspan, S., and Monty, M. (1983) "Evaluation and analysis of users' activity organization" *Proc ACM SIGCHI '83 Human Factors in Computing Systems* pp 54-57, Boston, December 12-15.

[Bann84] Bannon, L. and O'Malley, C. (1984) "Problems in evaluation of human-computer interfaces: a case study" *Interact '84 — First IFIP Conference on Human-Computer Interaction, 2* pp 280-284, London, UK, Sept 4-7.

[Burt81] Burton, R.R. and Brown, J.S. (1981) "An investigation of computer coaching for informal learning activities" in *Intelligent Tutoring Systems*, edited by D. Sleeman and J.S. Brown, pp 79-97, Academic Press.

[Drap84] Draper, S.W. (1984) "The nature of expertise in Unix" *Interact '84 - First IFIP Conference on Human-Computer Interaction, 2* pp 182-186, London, UK, Sept 4-7.

[Gree88] Greenberg, S., and Witten, I.H. (198) "How users repeat their actions on computers: Principles for the design of history mechanisms" *Proc ACM SIGCHI '88 Human Factors in Computing Systems,* Washington, D.C. May 15-19.

[Gree85] Greenberg, S., and Witten, I.H. (1985) "Interactive end-user creation of workbench hierarchies within a window interface" *Proc Canadian Information Processing Society National Conference*, Montreal, Quebec, June. Also available as Research Report 85/181/04, Dept of Computer Science, University of Calgary.

[Gree86] Greenberg, S., Peterson, M., and Witten, I. "Issues and experiences in the design of a window management system" *Proc Canadian Information Processing Society Edmonton Conference* pp 33-50, Edmonton, Alberta, October 21-23. Also available as Research Report 87/257/05, Dept of Computer Science, University of Calgary.

[Gree84] Greenberg, S. (1984) "User Modeling in Interactive Computer Systems" MSc Thesis, Dept of Computer Science, University of Calgary. Also available as Research Report 85/193/6.

[Hans84] Hanson, S.J., Kraut, R.E., and Farber, J.M. (1984) "Interface design and multivariate analysis of UNIX command use" *ACM Transactions on Office Information Systems, 2* (1), March.

[Joy80] "An introduction to the C shell" *Unix Programmer's Manual, Seventh Edition* Volume 2c, University of California, Berkeley

[Kern81] Kernighan, B.W. and Mashey, J.R. (1981) "The UNIX programming environment" *IEEE Computer 14* (4) pp 25-34, April.

[Knut73] Knuth, D.E. (1973) *The art of computer programming: searching and sorting.* Addison-Wesley.

[Kraut83] Kraut, R.E., Hanson, S.J., and Farber J.M. (1983) "Command use and interface design" *Proc ACM SIGCHI 83 Human Factors in Computing Systems*, pp 120-124, Boston, December 12-15.

[Niel86] Nielsen, J., Mack, R.L., Bergendorff, K.H., and Grischkowsky, N.L. (1986) "Integrated software usage in the professional work environment: evidence from questionnaires and interviews" *Proc ACM SIGCHI '86 Human Factors in Computing Systems*, 162-167, Boston, April 13-17.

[Peac82] Peachey, J.B., Bunt, R.B., and Colbourn, C.J. (1982) "Bradford-Zipf phenomena in computer system" *Proc Canadian Information Processing Society National Conference*, pp 155-161, Saskatoon, Saskatchewan, May.

[Perl84] Perlman, G. (1984) "Natural artificial languages: low-level processes" *Int J of Man-Machine Studies, 20* (4) pp 373-419

[Ritc74] Ritchie, D.M. and Thompson, K. (1974) "The UNIX time-sharing system" *Communications of the ACM, 17*, (7) pp 365-375.

[Slee82] Sleeman, D. and Brown, J.S. (1982) "Introduction: Intelligent tutoring systems" in *Intelligent Tutoring Systems*, edited by D. Sleeman and J.S. Brown, pp 1-8, Academic Press.

[Witt85] Witten, I.H. and Greenberg, S. (1985) "User interfaces for office systems" in *Oxford Surveys in Information Technology*, edited by P. Zorkoczy, pp 69-104, Oxford University Press.

[Witt84] Witten, I.H., Cleary, J., and Greenberg, S. (1984) "On frequency-based menu-splitting algorithms" *Int J of Man-Machine Studies, 21* (2), pp 135-148

[Zipf49] Zipf, G.K. (1949) *Human behaviour and the principle of least effort.* Addison-Wesley, Ontario.