How Users Repeat Their Actions on Computers: Principles for Design of History Mechanisms

Saul Greenberg and Ian H. Witten
Department of Computer Science
The University of Calgary,
Calgary, Alberta, Canada T2N 1N4

Abstract

Several striking characteristics of how often people repeat their actions on interactive systems are abstracted from usage data gleaned from many users of different classes over a period of months. Reformulated as empirically-based general principles, these provide design guidelines for history mechanisms specifically and modern user interfaces generally. Particular attention is paid to the repetition of command lines, and to the probability distribution of the next line given a sequential "history list" of previous ones. Several ways are examined of conditioning this distribution to enhance predictive power. A brief case study of actual use of a widely-used history system is also included.

Keywords: Command-based systems; command reuse; history mechanisms; human-computer interaction; design principles.

Introduction

Flexible interfaces create an environment in which users can pursue goals not considered specifically by any one application package. This paper addresses those top-level interfaces that provide a rich set of executable actions and objects. Actions are traditionally invoked by typing simple commands, although some modern systems augment or replace this primitive dialogue style with menus, forms, natural language, graphics, and so on [19]. Typically, these interfaces either provide uniform access to all system actions or group these actions in some pre-defined way.

Human usage of such computers is characterized by certain patterns of activity that are ill supported by contemporary interfaces. In particular, although it is well known that users often repeat actions, most systems do little to allow them to review and re-execute previous ones. Typically, they must be laboriously re-typed or re-selected through menu navigation. Those systems that do provide assistance offer *ad hoc* "history" mechanisms that employ a variety of recall strategies:

History through glass teletypes. Special textual syntactic constructs allow previous events to be recalled, usually by position on an event list (relative or absolute), or by pattern-matching. Examples are the UNIX *csh* [12] and the INTERLISP-D Programmer's Assistant [16].

History through graphical selection. A menu of previous events is presented which are manipulated graphically. HISTMENU[5] and MINIT 's "window management window" [4] are two examples.

History through editing. Any text appearing in the dialogue transcript can be copied to and edited further in the command input area. Examples include Apollo's DOMAIN window "pads" [1] and command interpreters running within the *emacs* editor [14].

History for menu navigation. Previously chosen menu items become more readily available than the default. The "bookmarks" capability of the Symbolics Document Examiner is one example [15]. Another is an adaptive algorithm that favourably relocates previously chosen items in a menu hierarchy [20], which has found success in an experimental telephone directory [9,17].

History through prediction. Within the current context, the system estimates for each token already seen the probability that it will be the next one typed. The one(s) with the highest probabilites are made available for selection (*eg* "Predict" [22] and the "Reactive Keyboard" [21]).

History through programming by example. Fixed sequences of actions are saved as a procedure, perhaps allowing some generalizations to be made [18].

Most history mechanisms are based on the simple premise that the last n user inputs are a reasonable working set of candidates for re-selection. But is this premise correct? Might other strategies work better? Indeed, is the dialogue sufficiently repetitive to warrant history mechanisms in the first place? As existing systems are designed through intuition rather than from empirical knowledge of user interactions, it is difficult to judge how effective they really are or what scope there is for improvement.

Sample Name	Sample Size	Recurrence Rate mean	Recurrence Rate std dev	Users of History actual	Users of History (%)	Mean rate of history uses (%)
Novice Programmers	55	80.4%	7.2	11/55	20%	2.03
Experienced Programmers	36	74.4%	9.7	33/36	92%	4.23
Computer Scientists	52	67.7%	8.2	37/52	71%	4.04
Non-Programmers	25	69.4%	8.1	9/25	36%	4.35
Total	168	73.8%	9.6	90/168	54%	3.89

Table 1: Sample sizes, recurrence rates and history uses of each group

This paper investigates user behavior relevant to the design of history mechanisms. The primary objective is to formulate general principles of how users repeat their actions on computers. The investigation is based upon analyzing long-term records of user-computer interaction with an imperative interface, collected as described in the following section. The research questions raised in the subsequent section help focus exploration of the large data set, and the results are analyzed from a variety of perspectives. A discussion follows in the last section, where specific principles are developed.

The UNIX *csh* command interpreter was used as a vehicle for this study, as it has been for many earlier investigations of how users interact with command-based interfaces. Its popularity makes it relatively easy to find and observe diverse sample groups of users in a realistic setting [1]. Although the command interface no longer represents current ideas in interface design, it is assumed that observed usage patterns are fundamental to similar computer-based imperative interactions. Studies of UNIX usage have already affected the design of leading-edge systems. For example, [6] described a multiple virtual-workspace interface to support user task switching, motivated by the UNIX study of [2].

Data Collection

Command-line data was collected continuously for four months from users of the Berkeley 4.2 UNIX *csh* command interpreter [12]. The start of every login session was noted, and all commands and arguments passed to *csh* were recorded sequentially. 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 (Table 1). Salient features of each group are described below.

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 fulfil 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 command lines entered by individual subjects ($mean$ = 1712,$std\ dev$ = 1499).

Data Analysis

Four questions particularly relevant to history mechanisms are addressed here. They all concern the statistics of complete command lines entered by the user, since history mechanisms usually involve the whole command line. First, we look at how often a user actually repeats command lines over the course of a dialogue. Second, we describe the probability distribution that the next command line will match a user's previous inputs by location in an event list. Third, since this distribution depends upon a simple model of arranging and matching the user's command history, alternative models are evaluated which condition the distribution in different ways. Finally, we note how people actually use the existing UNIX *csh* history facility.

[1] But see [3] for problems encountered even here.

In the following discussion, a *command line* is a single 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). 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 — duplicates are not pruned. The *distance* between two command lines is the difference between their positions on the list. A *working set* is a small subset of items on the history 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.

Recurrence of command lines

Most history mechanisms simplify redoing the complete command line, rather than its isolated components. Although it is known that only a small set of commands account for all user actions [2,7,8,11,13] [2], it is not known how often complete command lines recur. One might expect that they would not recur often, 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. Table 1 lists the mean recurrence rate and standard deviation for each subject group. An analysis of variance of raw scores rejects the null hypothesis that these means are equal ($F(3, 164) = 21.42, p < .01$). The Fisher PLSD multiple comparison test suggests that all differences between group means are significant ($p < .01$), excepting the Non-programmers versus Scientists. As the Table indicates, the mean recurrence rate for groups ranges between 68% and 80%, with Novice Programmers exhibiting the highest scores. Still, it is reasonable to approximate the recurrence rate by the population mean of 74%. That is, about three out of every four command lines entered by the user already exist on the history list. Conversely, an average of one out of every four appears for the first time.

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? If a graphical history mechanism displayed the previous n entries as a menu (*eg* HISTMENU [5]), what is the probability that this includes the next entry?

The recurrence distribution as a measure of distance was calculated for each user, and group means are plotted in Figure 1. The vertical axis represents the rate of command line recurrences, while the horizontal axis shows the position of the repeated command 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 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 1, which plots the same data for the grouped total 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 1 also shows that the differing recurrence rate between user groups, noted previously in Table 1, 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.

Conditioning the distribution

The recurrence distributions were derived by considering all user input as one long sequential stream, with no barriers placed between sessions. We have seen that a small local working set of command lines accounts for a high portion of repetitions. Consider a working set of the seven previous items on the history list. From the inset in Figure 1, there is a 26% chance that the next command line has not appeared before, a 43% chance that it has recurred within the working set, and a 31% chance that it last appeared further back. This section explores the possibility that the distribution can be conditioned to increase the recurrence probabilities. Three conditioning techniques are discussed: context sensitivity by directory; pruning repetitions; and par-

[2] This aspect of our study is reported in greater detail in a companion paper, which includes a discussion on individual differences in command selection and use [8]

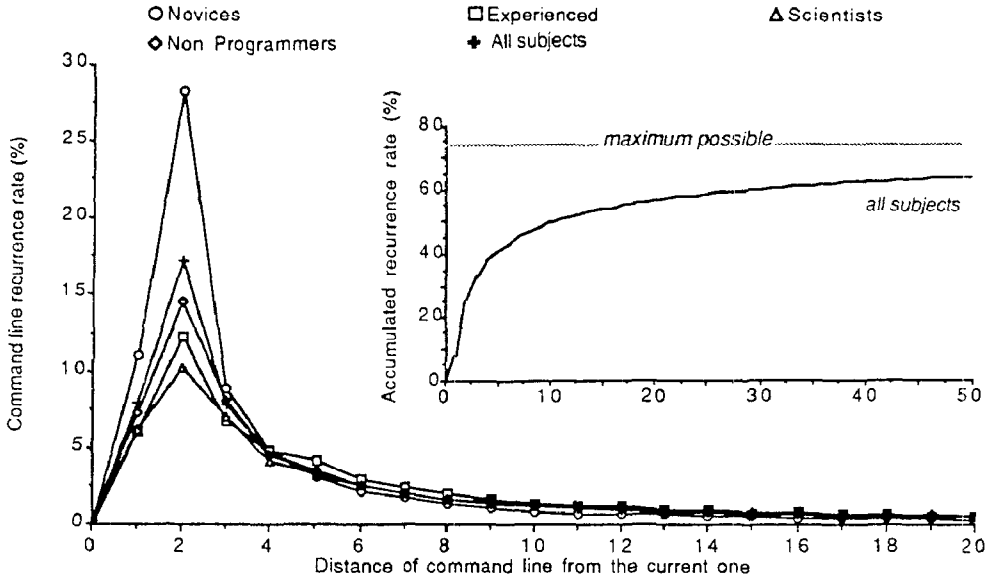

Figure 1: Recurrence distribution as a measure of distance

tial matches. An example of an event list altered by several conditions is shown in Table 2, where the first column represents the complete input sequence.

Directory-sensitive history lists. Users of computer systems perform much task switching [2]. Since many commands are specific to the task at hand, it is reasonable to assume that context-sensitive history lists will give better local predictions. UNIX provides a hierarchical directory system for maintaining files. As many user actions reference these files, we hypothesize that the current working directory defines a context for command lines. [3] This is tested by contrasting the recurrence distribution for directory-sensitive history lists with the standard sequential list. The second main column of Table 2 illustrates the directory-sensitive condition, where each sub-column is sensitive to a particular directory. [4] Most command lines here refer to files in the directory, and are not useful outside the directory. Some command lines, however, are common to both.

Pruning repetitions from the history list. The history lists mentioned so far maintain a record of every single command line typed. Duplicates are not pruned off the list. On a history list of limited length, duplicates occupy space which could more fruitfully be used by other command lines. There are two obvious strategies for pruning redundancies, as described by [4]. The first saves the command line in its original position on the history list while the second saves it in its latest position (Table 2). The latter was selected for study, as not only is local context maintained, but unique and low probability entries will migrate to the back of the list over time.

Partial Matches. Instead of the next command line matching a previous one exactly, partial matching may be allowed. This is helpful when people make simple spelling mistakes, the same command and options are invoked on different arguments, command lines are extended, and so on. However, the benefit is highly user-dependent, for the selected sequence must be altered before it is invoked. We investigated partial matches by prefix, where the matched command line is a prefix of the next command line, up to and including a complete match.

Combinations. The strategies above are not mutually exclusive, and all can be combined in a variety of ways. The bottom half of columns 2 and 3 of Table 2 shows one such possibility, where the event list is conditioned by directory sensitivity and pruning.

Data from the Experienced Programmers subject group, each of whom used more than one directory, was re-analyzed by applying the above conditions to the traces. The cumulative probability distributions of all conditions and their combinations are illustrated graphically in Figure 2.

Creating context-sensitive directory lists decreases the overall recurrence rate from 74% to 65%, as command lines entered in one directory are no longer available in others. Although this reduction means that plain sequential lists out-perform directory-sensitive ones over all previous entries, benefits were observed over small

[3] Properly associating a user's commands with their tasks or goals is not easy. We recognize that grouping commands by the current directories (or perhaps by the obvious alternative of windows) is just an estimate — possibly a poor one — of actual task contexts.

[4] In Unix, users change directories through the *cd* command. The "~" is shorthand for the home directory. Following "/"'s indicate sub-directories.

Sequential *starting in ~/text*		Directory Sensitive *directory context is ~/text*		Directory Sensitive *directory context is ~/figures*		Duplicates Removed *original position*		Duplicates Removed *latest position*	
1	ls	1	ls	6	ls	1	ls	4	edit draft
2	edit draft	2	edit draft	7	edit fig1	2	edit draft	8	edit fig2
3	print draft	3	print draft	8	edit fig2	3	print draft	9	graph fig1
4	edit draft	4	edit draft	9	graph fig1	5	cd ~/figures	10	ls
5	cd ~/figures	5	cd ~/figures	10	ls	7	edit fig1	11	edit fig1
6	ls	13	print draft	11	edit fig1	8	edit fig2	12	cd ~/text
7	edit fig1	14	cd ~/figures	12	cd ~/text	9	graph fig1	13	print draft
8	edit fig2					12	cd ~/text	14	cd ~/figures
9	graph fig1	*with duplicates removed, events saved in latest position*							
10	ls	1	ls	8	edit fig2				
11	edit fig1	4	edit draft	9	graph fig1				
12	cd ~/text	13	print draft	10	ls				
13	print draft	14	cd ~/figures	11	edit fig1				
14	cd ~/figures			12	cd ~/text				

Table 2: Four examples of a conditioned event list

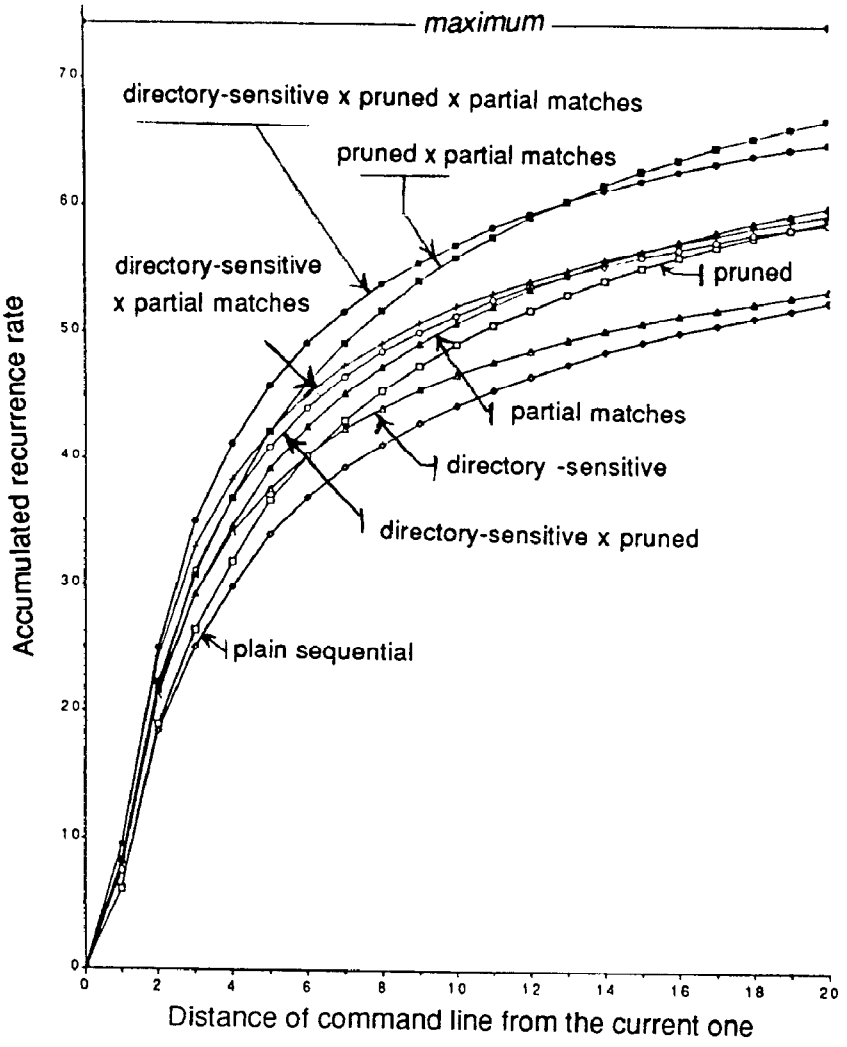

Figure 2: Conditioning the probability distribution

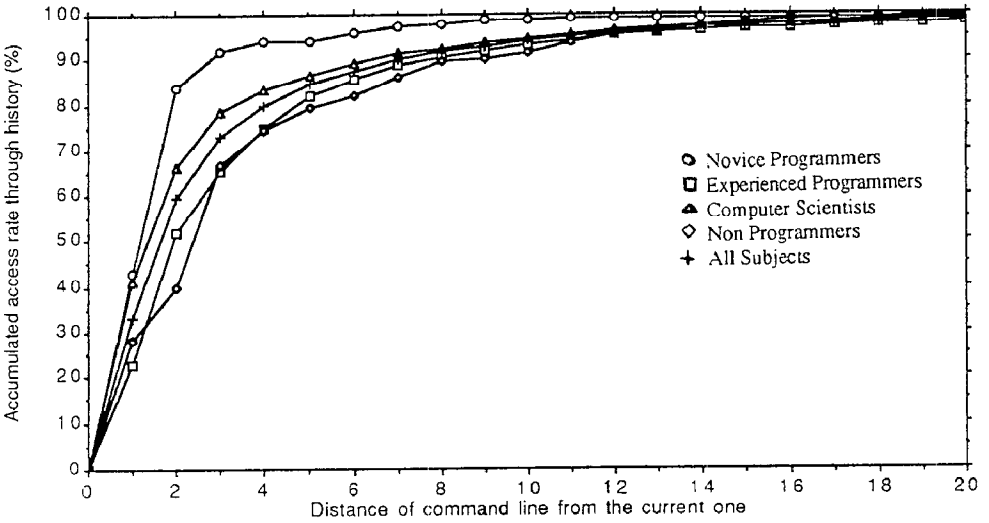

Figure 3: Accumulated distribution of history use as a measure of distance

working sets. The first three directory-sensitive items are more probable than their equivalent sequential items, approximately equal for the fourth, and slightly less likely thereafter. With a working set of ten items, directory sensitivity increases the overall probability of the working set by 2.5%.

Although pruning duplicates off the history list does not alter the recurrence rate, it does shorten the total distance covered by the distribution. As the working set size increases, so do the accumulated probabilities when compared to the standard sequential list (Figure 2). Pruning duplicates increases the overall probability of a ten-item working set by 5%.

Pattern matching by prefixes increases the recurrence rate to 84%. [5] As partial matches are found before more distant (and perhaps non-existent) exact matches, an increase is expected in the rate of growth of the cumulative probability distribution. This increase is illustrated in Figure 2. Conditioning by partial matching increases the overall probability of a ten-item working set by around 6%.

When conditioning methods are combined, the effects are slightly less than additive. Figure 2 illustrates these combinations. For example, a partially-matched, pruned and directory sensitive history mechanism out-performs a plain sequential one by 13% with a working set of ten items.

Actual use of Unix history

We have seen that user dialogues are highly repetitive and that the last few command lines have the greatest chance of recurring — the premise behind most history systems. But are current history mechanisms used well in practice? We investigated this by analyzing each user's *csh* history use.

[5] In this context, the recurrence rate is the probability that any previous event is a prefix of the current command line.

The recurrence rate and its probability distribution, studied previously, provides a value against which to assess how well history mechanisms are used in practice. The average rate of re-selecting items through history cannot exceed the recurrence rate, which was found to be 74%. By comparing the user's actual re-selection rate when using a particular history mechanism with this maximum, the system's practical effectiveness can be judged.

Table 1 shows how many users of UNIX *csh* in each sample group actually used history. Although 54% of all users recalled at least one previous action, this figure is dominated by the computer sophisticates. Only 20% of Novice Programmers and 36% of Non-Programmers used history, compared to 71% for Computer Scientists and 92% for Experienced Programmers.

Those who made use of history did so rarely. On average, 3.9% of command lines referred to an item through history, although there was great variation ($std\ dev$ = 3.8;$range$ = 0.05% – – 17.5%). This average rate varied slightly across groups, as illustrated by the last column in Table 1, but an analysis of variance indicated that differences are not statistically significant ($F(3, 86) = 1.02$).

In practice, users did not normally refer very far back in history. With the exception of novices, an average of 79 – 86% of all history uses referred to the last five command lines. Novice Programmers achieved this range within the last two command lines. Figure 3 illustrates the nearsighted view into the past. Each line is the running sum of the percent of history use accounted for (the vertical axis) when matched against the distance back in the command line sequence (the horizontal axis). The differences between groups for the last few actions reflect how far back each prefers to see. Although most uses of history recall the last or second last entry, it is unclear which is referred to more.

It was also noticed that history was generally used to access or slightly modify the same small set of command lines repeatedly within a login session. If history was used to recall a command line, it was highly probable that subsequent history recalls will be to the same command.

Subjects indicated that they are discouraged from using *csh* by its difficult syntax, the incomprehensible manual entry, and the fact that previous events are not normally kept on display. Also, the typing overhead necessary to specify all but the simplest retrievals makes them feel that it is not worth the bother.

Discussion

Our analyses of command line recurrences within the UNIX *csh* dialogue produced specific results in several areas. Based on these results we formulate some empirically-based general principles of how users repeat their actions on computers.

1. **A substantial portion of each user's previous actions are repeated.** In spite of the large number of options and arguments that could qualify a command, command lines are repeated surprisingly often.
2. **New command lines are composed regularly.** Although many actions are repeated, a sizeable proportion are new.
3. **Users exhibit considerable recency.** The major contributions to the recurrence distribution are provided by the last few command lines entered, independent of context.
4. **Some actions remain outside the local working set.** A significant number of recurring items are not covered by the last few items. Doubling or even tripling the size of the working set does not increase the coverage significantly.
5. **Working sets can be improved by suitable conditioning.** A perfect "history oracle" would always predict the next user command line correctly, if it was a repeat of a previous one. As no such oracle exists, we can only contemplate and evaluate methods that offer the user reasonable candidates for re-selection. Although simply looking at the previous n user actions is reasonably effective, context sensitivity, pruning duplicates and partial matches increase coverage to some degree. [6]
6. **When using history, users continually recall the same command lines.** UNIX users generally use history for recalling the same events within a log-in session.
7. **Unix *csh* history does poorly.** Most people (especially novices and non-programmers) don't use it. Those who do, don't use it much.

General design guidelines are self-evident from these principles. Once the style of interface is specified, the guidelines formed could become much more specific. For example, if a menu of the previous n items are to be displayed, and no user data is available, the best value of n could be estimated from the recurrence distributions shown in this paper. Similarly, the complexity required by syntactic constructs used to retrieve command lines in glass-teletype history mechanisms can be judged (*ie* constructs retrieving probable command lines should be simple). Or perhaps context conditioning for window-based interfaces are defined by window context, rather than by directory. It is beyond the scope of this short paper to discuss all possibilities.

Conclusions

This paper has set out empirically-justified principles of how people repeat command lines, and indicated that the high recurrence rate observed justifies the inclusion of history mechanisms to certain user interfaces. Using these principles, designers now have a basis for evaluating and fine-tuning existing history mechanisms, or creating new ones.

There are still many unanswered questions. We have not formed any hypotheses of *why* users repeat their actions the way they do. Nor do we know how generalizable our results are. We are now in the process of extending this investigation, both through further analysis and through applying our results to the design and implementation of a window-based history mechanism, and are working towards integrating history with task-oriented workspaces [10].

References

[1] Apollo (1986) "DOMAIN System User's Guide" Apollo Computer Inc, Chelmsford, Mass

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

[3] 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* 280-284, London, UK, Sept 4-7.

[4] Barnes, D.J. and Bovey, J.D. (1986) "Managing command submission in a multiple-window environment" *Software Engineering Journal, 1* (5) 177-183, September

[6] But conditioning strategies are not always appropriate. Pruning, for example, would interfere with the undo capabilities provided by some systems (*eg* [5]).

[5] Bobrow, D.G. (1986) "HistMenu" in *Lisp User Library Packages Manual, Koto Release*, Xerox Artificial Intelligence Systems, 215-216, April

[6] Card, S.K. and Henderson Jr., A (1987) "A multiple, virtual-workspace interface to support user task switching" *in Proceedings of ACM CHI+GI 1987 Human Factors in Computing Systems and Graphics Interface* 53-59, Toronto, April 5-9.

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

[8] Greenberg, S. and Witten, I.H. (1988) "Directing the user interface: how people use command-based systems" *in Proceedings of the 3rd IFAC Conference on Man-Machine Systems* Oulu, Finland, June 14–16 (in press).

[9] Greenberg, S. and Witten, I.H. (1985) "Adaptive personalized interfaces — a question of viability" *Behaviour and Information Technology, 4* (1) 31–45.

[10] 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

[11] 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.

[12] Joy, W. (1980) "An introduction to the C shell" in Unix Programmer's Manual, Seventh Edition, Volume 2c University of California, Berkely, California

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

[14] Stallman, R.M. (1981) "EMACS the extensible, customizable self-documenting display editor" *ACM Sigplan Notices — Proceedings of the ACM sigplan SIGOA symposium on text manipulation, 16* (6), 147-155

[15] Symbolics (1985) "Using the online documentation system" in *User's Guide to Symbolics Computers* Symbolics, Inc.

[16] Teitelman, W., Masinter, L. (1981) "The Interlisp programming environment" *IEEE Computer, 14* (4), 25-34

[17] Trevallyan, R., and Browne, D.P. (1987) "A self-regulating adaptive system" *in Proceedings of ACM CHI+GI 1987 Human Factors in Computing Systems and Graphics Interface* 103-107, Toronto, April 5-9.

[18] Witten, I.H. MacDonald, B.A., and Greenberg, S. (1987) "Specifying procedures to office systems" *Automating Systems Development Conference*, Leicester, April 14-16

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

[20] Witten, I.H., Cleary, J., and Greenberg, S. (1984) "On frequency-based menu-splitting algorithms" *Int J Man-Machine Stduies, 21* (2) 135–148, August.

[21] Witten, I.H., Cleary, J.G., and Darragh, J.J. (1983) "The reactive keyboard: A new technology for text entry" *Proc Canadian Information Processing Conference* 151-156, Ottawa, Ontario, May

[22] Witten, I.H. (1982) "An interactive computer terminal interface which predicts user entries" *Proc IEE Conference on Man-machine Interaction* 1-5, Manchester, England, July