Chapter 4

User Interaction Behaviour and History Usage

The preceding analysis of the seven potential uses of history shed light on a number of cognitive and behavioural considerations underlying the use of history tools. This analysis permits a broad examination of the scope and coverage of these relevant variables and hence, user requirements. However, the insights are based on an informal task-artifact analysis of current history tools rather than a behavioural analysis of user interactions and history tool use. Behavioural analysis of computer-tool usage provides another source of information about user requirements; one that is directly supported by behavioural data. However, it is conducted within a specific and focussed context and is limited in the issues that can be examined and the generalizations that can be made.

This chapter describes an exploratory behavioural study of three UNIX command interpreters and their history tools. It is exploratory because everyday natural user interactions within UNIX are observed, unconstrained by experimental controls. The motivation for performing this particular study is presented, followed by a description of the study and its findings.

Behavioural Studies of User Interactions and History Use

Several studies have investigated how individuals and groups of individuals interact with computer systems. Most of them concentrate on command usage and makeup of command sets to derive general implications for user interface design (e.g., Akin, Baykan, & Rao, 1987; Boies, 1974; Draper, 1984; Greenberg & Witten, 1988a; Hanson, Kraut, & Farber, 1984; Krishnamurthy, 1987). They identify important behavioural trends regarding frequency of command use or growth in command sets. However, very few studies have focused on uncharacteristic user interaction behaviours arising from constraints in computer tools and environments (i.e., behavioural phenomena). An example of such a study is Greenberg and Witten (1988b)'s study which found that users frequently issued previous command lines (i.e., on average, 74% of the commands lines are recurrences). We are interested in such behavioural phenomena because they can provide insights into the kinds of user support tools, particularly history tools, that we need to provide in order to nurture and enhance user interactions.

Furthermore, in order to shed light on how history tools can provide user support for user interactions, we need to examine the kinds of functions in history tools that people use. We are aware of only one study that has examined actual usage of history tools (i.e., Greenberg and Witten (1988b)'s study). That study documented the extent of history usage (54% of the subjects used history but in only 4% of their commands) and the referenced history items (the bulk of the references were to the last 2 commands), but their study did not examine the history functions that were used. Such observations could reveal the strengths and weaknesses of history tools and the user needs and requirements with respect to the content of a user history and the history functions.

Exploratory Study of User Interactions and History Usage

Since previous studies have not examined certain behavioural aspects related to user interactions and history-tool usage, we conducted our own study. The study has two objectives. The first objective was to identify behavioural phenomena in everyday natural user interactions that are particularly relevant to history-based, user support tools. The second objective was to examine the history functions that users employ and to compare usage of two history tools.

The exploratory study observed everyday natural user interactions of users of three different UNIX command interpreters – Bourne Shell (*sh*) [Bourne, 1979], C Shell (*csh*) [Joy, 1980], and TC Shell (*tcsh*) [Ellis, Greer, Placeway, & Zachariassen, 1987] – with no experimental controls and no observer present. Three UNIX shells are examined because they provide a richer pool from which subjects can be drawn and from which similarities and differences in user interaction behaviours and history use can be observed. While this methodology may reveal natural behavioural patterns evident in user interactions, it does not provide information about the factors influencing the behaviour. Therefore, its findings should be interpreted with caution. Furthermore, UNIX is a textual, command-based system and the contents of a user's history in *csh* and *tcsh* are the command lines. Therefore, issues pertaining to graphical, gesture-based systems are not explored.

A possible criticism may be made about studying a system with poor usability such as UNIX when direct manipulation systems like the MACINTOSH are popular and widely used. The following reasons are offered in answer to such a criticism:

(1) There is a large user community with varying skills and task complexities.

(2) This study adds to the accumulating body of knowledge about user interactions in UNIX [Akin, Baykan, & Rao, 1987; Bannon, Cypher, Greenspan, & Monty, 1983; Desmarais & Pavel, 1987; Draper, 1984; Greenberg & Witten, 1988a; Greenberg & Witten, 1988b; Hanson, Kraut, & Farber, 1984].

(3) The shells permit user access to a wide range of UNIX applications. Hence, we can potentially observe general patterns of behaviour.

(4) Most UNIX shells support history tools (e.g., Korn Shell, C Shell, TC Shell).

While UNIX has some usability problems which may result in some artificial behaviours, these problems should not preclude the observation of behaviours that are fundamental to user interactions with textual command-based systems.

Method

UNIX Shells and History Tools

A *UNIX shell* is a programmable UNIX command interpreter. We selected *sh*, which does not support a history tool, because user behaviours in this shell can be compared against those behaviours observed in UNIX shells which support history tools (i.e., *csh* and *tcsh*). In *csh* and *tcsh*, command lines that are issued are collected into a *history list* which is displayed upon the user's explicit request. The history list is a user history and each of its entries contains a user command line known as an event.

A *csh* user command line may be a shell request (e.g., ls file), a history request (e.g., !!) or a hybrid request (e.g., !! | lpr -Pntx). A hybrid request combines shell and history requests. A *csh* history request has the following syntax:

event designators [**:** *word designators*] [**:** *modifiers*]

Optional items are delimited by the symbols []. *Event designators* select particular events in a history list (e.g., *!!* for the most recent history event). *Word designators* select words from the associated history event (e.g., *:2-4* for the 2nd, 3rd, and 4th word). *Modifiers* request that changes be made to the associated history event (e.g., *:1* extracts the second word of the associated event).

This syntax allows event designators to be bound to zero or more word designators and/or modifiers (e.g., *!-5:s/aaa/bbb* recalls the fifth last event and makes the substitution before submitting the modified history event). Thus, users can modify a history event before issuing it. *csh* provides a number of short cuts for word designators (e.g., !$ is the last word of a history event) and modifiers (e.g., ^pattern^replacement^ replaces portions of the last event matching *pattern* with *replacement*).

The command *history* prints a portion of a history. As well, when the *:p* modifier is used with a history event designator, the retrieved history event is printed rather than issued (e.g., !!:p prints the last history event). Table 1 lists the number of event designators, word designators, modifiers, and history print requests in *csh*. Note that the number of modifiers excludes the *:p* modifier which is accounted for in the number of history print requests.

tcsh is a superset of *csh* and supports two enhancements to the *csh* history tool: a) a history retrieval operation may be performed separately from a history modification operation and b) command line editing and history manipulations are facilitated by emacs-style control keys.

Event Designators	Word Designators	Modifiers	List Contents of History	Total Number of Operators
7	11(4)	9(2)	2	35

Table 1: The number of *csh* history functions in each category. The number in a parenthesis represents the number of short cuts.

Two history retrieval options are available in *tcsh* :

a) step backwards and forwards through a history list examining each event one at a time, using a cursor-like control key.

b) provide a pattern followed by a search backwards/forwards control key. The system searches backwards or forwards through the history list for the first occurrence of a history event beginning with the characters in the pattern.

Both history retrieval options and edit operations are invoked from the user's current input location in the shell session with the retrieved history event overwriting the history retrieval request. Effectively, a one-line edit buffer exists at the user's current input location.

Data Collection

The Berkeley 4.2BSD *script* program records everything printed in a user session. When it is invoked, its process is inserted between the shell process and the terminal process. The terminal process controls user inputs from the terminal's keyboard and system outputs for the terminal.

The *script* program was modified to produce a *server* and *client* version of the program. There are two reasons for doing this rather than modifying the three shells. First, less effort is required to instrument one program than to instrument three UNIX command interpreters to log the usage data. Second, using this approach, it is possible to identify when users invoked history functions, what sorts of history functions they made use of, and what sorts of command line editing were performed. Logging the data after the user's input is parsed by the shell precludes access to the history usage data.

In the *server-client* version, the *server-script* process rendezvous with one or more *client-script* processes dispersed, possibly, on different machines to record the data collected by the *client script* program into a central file. Each *client-script* process is associated with a user session and records the raw keystrokes (e.g., vi ^H !$) initiated in that user session.

In general, a shell process parses the raw keystrokes to produce the fully-parsed command line. If we only capture raw keystrokes, we have no access to the fully-parsed command line. We can interpret some of these raw keystrokes (e.g., the raw keystrokes *cd^H^Hls News* can be easily interpreted as *ls News* because ^H is the control character for backspace). However, we are unable to interpret all raw keystrokes. For example, we cannot interpret the ^[character in the raw keystroke *cd News/^[* without recording information about the contents of the directory from which this command was issued because the control character ^[completes the file name with the name of a file in the current directory matching the starting pattern.

Since we cannot interpret all raw keystrokes without recording extraneous information, we use a heuristic to capture fully-parsed command lines that the shell echoes as it parses raw keystrokes. This heuristic scans the first n characters of output lines from the shell process for the user's prompt character c. The two parameters, c and n, are set up individually in consultation with each subject at the beginning. We had mixed success using this heuristic to record every fully-parsed command line. Consequently, we did not manage to record all fully-parsed

command lines matching all recorded keystrokes. While this flaw is a drawback, it is not a serious one because we are able to observe, qualitatively, some behavioural phenomena based on the fully-parsed command lines that we did record. However, we are unable to quantify the extent of any observed behavioural phenomena (e.g., mean number of recurrence of the same fully-parsed command line).

Subjects

6 *sh*, 10 *csh*, and 7 *tcsh* users volunteered for the study. They were graduate students, staff, and faculty members from computing sites on campus who were skilled users of the particular shell. They used UNIX to carry out their day-to-day tasks (e.g., system administration, programming, statistical analysis, document preparation and communication).

Knowledge and use of the history tool were not prerequisites for participation in the experiment as the study's objectives were to get a representative sample of users. If such prerequisites had been used, the study would have ignored the behavioural characteristics of non-history-tool users and we would not have gotten a sense of the number of non-history-tool users.

In the analysis, only the data for those subjects who issued 500 command lines or more during the monitoring period were examined. This restriction eliminated those subjects who decided to withdraw from the study and allowed us to observe behavioural patterns in UNIX user interactions. In the latter case, any behavioural patterns existing in user interactions would be more evident in sessions where many user command lines were issued. As a result, only the traces for 3 of the 6 *sh* (S1-S3), 5 of the 10 *csh* (S4-S8), and 5 of the 7 *tcsh* subjects (S9-S13) were analyzed.

Procedure

Each subject ran the client-script program in each of their shell sessions. The program timestamps and transmits all the user's raw keystrokes to the *server-script* process. Subjects were monitored in their natural work setting over a one week period with no experimenter present. The following data were collected: 1) the user's raw command line keystrokes, 2) the parsed shell command line, and 3) the time that it took the user to issue the command line.

Results

The findings are divided into three parts. Part one discusses a basic property evident in all the subjects' command lines and parts two and three discuss the history functions that *csh* and *tcsh* subjects used.

Three Behavioural Patterns

There are three patterns evident in our subjects' traces (see Table 2):

- Users *repeat* individual command lines or their parameters.
- Users *cycle* through one or more command lines.
- Users *group* command lines issuing them in no specific order.

As indicated earlier, because we are unable to capture all fully-parsed UNIX command lines, we are unable to characterize the frequency of each behavioural pattern (i.e., fraction of the session where each pattern was observed). However, this problem simply means that we cannot provide this statistic.

Parameters of command lines are not the same when they are re-issued. Observations and user comments suggest that these patterns are the result of task and user factors. Certain tasks are inherently repetitive while others are complex and ill-defined and require a trial and error approach. Users make errors and must make corrections and adjustments to their actions (e.g., misspelling or mistyping the command line) or they are unfamiliar with the system or the task.

History-Tool Functions Used by csh Subjects

Table 3 summarizes the extent to which *csh* and *tcsh* subjects, respectively, used the various functions available in their history tools. Each figure in the table

Description of Pattern	Example
repeating command lines and parameters literally	spell paper.tex | page spell paper.tex | page
repeating command lines with a different parameter	rlogin utworm rlogin utbugs
repeating with the same parameter but a different command name	page draw vi draw
cycling through multiple command lines with different parameters	setenv TERM xterm gnuemacs –e mh-r setenv TERM vt100 gnuemacs –e mh-r setenv TERM sun gnuemacs –e mh-r
grouping command lines and executing them together	S < draw > draw.out page draw.out page draw.out vi draw S < draw > draw.out vi draw

Table 2: Examples of repeating, cycling, and grouping.

represents the percentage of the command lines the subject issued during the monitoring period that were issued using a particular history function [1]. All *csh* and *tcsh* subjects, including those who do not provide a long enough trace, made some use of the history tools.

Subjects did not, in practice, use the full range of functions available in their history tools (see Table 3) because they had limited knowledge of the repertoire of history functions. Furthermore, only a small fraction of the subject's session involved history-tool use. Subject S9 used a history tool the most (14%) while subject S8 used it the least (3%). Since we did not have access to all fully-parsed command lines, we are unable to compare the number of command lines that were literal repetition of earlier command lines against the actual number of command lines formed using history tools. However, this simply means that we cannot report this statistic.

csh subjects tend to use simple history functions involving literal recall of a history event (i.e., *!!* for last event and *!pattern* for most recent event beginning

History-Tool Function Used	Usage (% command lines)									
	csh Subjects					*tcsh* Subjects				
	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13
csh event designators										
!!	1.62	2.28	0.26	0.39	0.70	0.29	0.59	-	0.19	-
!pattern	3.28	5.52	6.78	2.42	0.53	0.75	2.58	3.06	6.74	0.63
!number	1.18	-	-	-	-	-	-	-	0.19	-
!-number	1.36	0.24	-	-	-	-	-	-	0.19	-
!?pattern	0.39	-	-	0.06	-	-	-	-	-	-
csh word designators										
!$	3.81	2.16	1.92	0.29	1.23	1.38	-	0.74	-	0.32
csh modifiers										
!! + add on	1.09	1.32	1.79	0.35	-	0.12	-	0.37	0.19	0.63
:gs/aa/bb, :s/aa/bb	0.07	-	-	0.04	-	-	-	-	0.10	0.16
^pattern^replacement^	0.96	0.36	0.13	0.33	-	0.12	-	-	0.58	-
csh requests										
list history	-	1.75	-	.16	.18	1.21	-	-	0.96	-
tcsh enhancements										
step back/forth						8.19	9.61	5.76	0.19	2.22
search back/forth						2.14	0.20	0.25	-	0.63
Total	13.76	13.63	10.88	4.04	2.64	14.20	12.98	10.18	9.33	4.59

Table 3: History-tool functions used by *csh* and *tcsh* subjects.

[1] Our figures are larger than Greenberg and Witten (1988b)'s figures because they may reflect situations in which a command line is composed using a number of history functions. Thus, the reader should be cautious about making direct comparisons between the figures from the two studies.

with the given pattern). Some subjects did make minor use of other event designators: *!n* – an absolute reference to the *nth* history event, *!-n* – a backward relative reference to the nth history event, and *!?pattern* – a reference to the most recent history event containing a given pattern.

In a number of cases, the subjects included words from a history event in composing their command lines (e.g., *string * | !bm* takes the output of *string ** as input to a previous command line beginning with the command *bm*). The principal reference was the short-cut word designator for the last word of an event (i.e., *!$*). A few of the subjects made rare use of the other *word designator* functions: both the regular form (i.e., *:1*) and the short-cut form (i.e., *!**).

A few subjects opted to make modifications to a history event before issuing it (see the *csh* usage figures for the *csh* modifier in Table 3). In many of these cases, subjects either added text to the end of the recalled event (e.g., *!! -l* invokes the last event with a *-l* option) or changed the previous event using the short-cut substitution operation (i.e., *^pattern^replacement^*). Some subjects did make use of substitution requests available in the *:modifier* function; they included the *:s/aa/bb* request which replaces an occurrence of aa with bb in the associated event and the *:gs/aa/bb* request which replaces all occurrences of *aa* with *bb* in each word of an event.

Finally, only 2 of the 5 *csh* subjects used the history print function.

History-Tool Functions Used by tcsh Subjects

tcsh subjects made slightly greater use of history tools compared to *csh* subjects. In general, they preferred to use *tcsh*'s visual history *step back/forth* rather than *csh's event designators* to recall a history event. However, most *tcsh* subjects opted to make use of *csh*'s method of recalling an event when they were searching with a pattern (i.e., *!pattern*) rather than the equivalent *tcsh* request, *search back/forth* .

tcsh subjects made less use of *csh* event designators and word designators; in general, they used the same simple functions that were used by *csh* subjects. *tcsh* subjects preferred to use control keys to edit command lines. The two rows of figures in the *tcsh enhancements* section of Table 3 bundle the cases when *tcsh* subjects simply recalled a command line and when they recalled a command line and modified it with control keys. However, *tcsh* subjects did make modifications when they recalled a history event which is unlike the *csh* subjects who tended to recall the history events literally. While modifications were performed, they were very simple ones. Typically, *tcsh* subjects altered a few characters at the end of the recalled command line or altered a word in the command line. *tcsh* subjects did not make changes to more than one word of the recalled command line.

Discussion

Our study is only one of two studies to examine the nature of history tool usage; the other was by Greenberg and Witten (1988b). While their study provides a number of useful findings and design guidelines, we need further elaboration, corroboration, and investigation of history tool usage. Also, our study examines

the types of history functions that are used, in practice, while their study quantified the extent to which history tools are used without analysis of the functions used.

The findings should be interpreted with caution because the study is not controlled, the subject pool is not large, and only command-based interfaces are studied. However, there are several implications for user requirements.

First, the observed patterns – systematic repetition of whole or parts of one or more command lines – and the nature of history tool usage provide evidence of reuse of whole command lines as well as partial reuse of commands and arguments. Thus, the ability to refer to components of a history event is desirable.

Second, subjects made changes to some recalled command lines before issuing them. The changes were made with substitution requests available in *csh* and emacs-like edit commands in *tcsh*. Thus, the ability to make changes to recalled command lines is desirable.

Third, subjects used a small fraction of the 35 different history functions available in *csh*. This observation is explained, in part, by the subjects' limited knowledge of the history functions. Thus, user training is desirable.

Fourth, subjects did not always use history tools to re-issue previous command lines. Greenberg and Witten (1988b) suggest that this is because the syntax for recalling a *csh* history event is obscure and arcane. This is substantiated by the fact that, although the *csh* history functions are available to *tcsh* subjects, 4 of the 5 *tcsh* subjects prefer to use the *tcsh* enhancements (see Table 3). These enhancements permit users to manipulate history events directly and this function may be more desirable than all the *csh* functions for recalling and modifying history events. *csh* provides an abstract and symbolic means of selecting and editing history events (e.g., use of regular expressions and pattern matching) compared to the direct interaction with history events that *tcsh* provides.

Fifth, the lack of a permanent display of a user history may be another plausible reason for the minimal use of history tools. Visual presentation of history events is important both for refreshing a user's memory and for allowing a user to consult the history. Neither history tool provides a permanent display of portions of a user history except upon explicit user request. The history usage data shows that both groups of subjects make very few explicit requests to display events in a history list (see Table 3). As such, *csh* subjects must rely heavily on their memory of recent history events. The task of recalling a history event in the absence of any external support from the display is memory-intensive and users may opt to selectively remember and recall memorable history events (e.g., recent command lines). In contrast, *tcsh* users are able, at least, to peruse history events as they single step through the history list.

Concluding Rem arks

There are two observations from the exploratory study. First, there are three user interaction patterns: repeat individual command lines or their parameters, cycle through one or more command lines, and repeat several command lines in no specific order. These patterns reveal a property evident in all the subjects' traces; subjects repeat command lines, in whole and in part, as a single directive and as a

group of directives. While Greenberg and Witten (1988b) observed that individual command lines are repeated, our study observed periods in a user's session in which the repetition is made with respect to a collection of command lines (see Table 2). The extent to which this property occurs in a subject's session is not quantified because it was not always possible to deduce the complete command line that users issued. In the next chapter, this behavioural phenomenon is detected mechanically and the extent to which it occurs is characterized for a different but larger set of trace data, using the concepts of locality and working set from computer memory research.

Second, some of the commands lines that the subjects re-issued were submitted with the aid of a small subset of the history functions. Of the *csh* history functions used, the bulk of them were simple ones such as the literal recall of a history event or the use of short-form history functions. By and large, *tcsh* subjects recalled history events by stepping or searching back and forth through their history and rarely made use of the *csh* event designator functions. However, unlike *csh* subjects, *tcsh* subjects edited command lines retrieved from their history list, changing a few characters or a word in the history event.

These two observations suggest three user requirements. First, users need better system support to facilitate the repetition of previous command lines (i.e., a history tool for reuse). Second, users need to be able to refer to components of a history event, to make changes to the recalled history event, and to display their history list. Third, users need to be aware of the availability of history tools and their functions.

In light of the limited set of capabilities available in current history tools (see discussion in Chapters 2 & 3), it is not surprising to discover that subjects do not make more extensive use of history tools when the opportunity arises. Besides poor design and poor user knowledge of history functions, there are two other factors that can influence people's use of history tools: availability of suitable reuse candidates and user effort required to use history tools.

The first factor is the ability to characterize relevant history events and to predict opportunities for using history tools. One such type of opportunity is suggested by the three command-line reference behaviours observed our exploratory study. The next chapter is a systematic examination of the three behavioural patterns and the implications of these reference behaviours on the prediction of history-for-reuse opportunities and history-for-reuse candidates.

The second factor is the mental and physical effort associated with the use of history tools. History tools may impose undue mental effort as compared to the physical effort required to retype a command line. As indicated earlier in Chapter 1, users want to minimize such mental effort. The tradeoff between mental and physical effort associated with using or not using history tools is examined in Chapter 6.