



The University of Sydney

A toolkit for appraising the long term usability of a text editor

Technical Report Number 489

October 1994

Ronny Cook, Judy Kay,
Greg Ryan and Richard C Thomas

ISBN 0 86758 942 6

**Basser Department of Computer Science
University of Sydney NSW 2006**

A toolkit for appraising the long term usability of a text editor

Ronny Cook, Judy Kay, Greg Ryan, Richard C Thomas†

Basser Department of Computer Science,
University of Sydney, 2006 Australia

† Department of Computer Science,
The University of Western Australia, Nedlands, WA 6009, Australia

Abstract

We describe a large scale, low cost, project that has examined the way people develop their skill in using fundamental software tools. The study involved over two thousand users during a three-year period of use of the sam text editor. The work took place while the editor was being employed in normal day to day work - it was not a laboratory experiment.

Our main contributions are first to demonstrate very long-term, low-cost monitoring with collections of simple analysis tools. Second, we have started to develop an understanding of how usability changes in the long term. Third, studies of usability often concentrate on assessment before a system is released for widespread use, whereas ours can help inform the long term design of new tools - a different dimension of usability. In addition we have mixed snap-shot studies with descriptions of long-term, gradual change. We can track the full development of the user, even though the quality of the data is lower than that normally associated with usability studies.

1. INTRODUCTION

The most highly skilled individuals have usually acquired their distinctive level of expertise over years or even decades, be they cathedral stone-masons or medics. As the computer becomes all-pervasive, it will be essential not to inhibit this long term self-development of skill. It is important, therefore, to understand the usability of fundamental software tools. One of the most basic of such tools supports users in creating and modifying texts: a text editor. Accordingly, we believe that learnability and usability testing of text editors, over the long term, is important.

There have been many studies of the ways that users learn about and actually employ text editors. This is natural given their importance. For example, see Roberts and Moran [25], Allwood and Eliasson [1], Kay and Black [15], Mack, Lewis and Carroll [19], and Sebrechts, Marsh and Furstenburg [27]. There have also been many studies of the usability of various text editing tools, for example: Rosson [26], Poller and Garter [23] and surveys such as Bosser [3].

Usability has several components. Classically, it is viewed in terms such as those suggested by Shackel [28] and Chapanis [5] with factors like: *effectiveness*; *flexibility*; *ease of use* and *learnability*; and the user's *attitude* to it. They assess these aspects of usability for a given user population, fixed tasks, fixed support and a fixed environment. This definition, and much of the work in the literature cited above, leans towards a view of usability that can be assessed in laboratory settings.

This view of usability is not yet generally agreed, nor will consensus necessarily emerge in the near future [2]. There are still problems, for example, in the term flexibility. On the one hand, Booth suggests a “system must be flexible, in that user performance must not deteriorate by more than a certain percentage across tasks and environments”. On the other Lindgaard [18] prefers to see it as referring "to variations in task-completion strategies supported by the system", and draws attention to the trade offs between learning and complexity that such variation entails.

Most research on usability of editors has studied small numbers of users, often quite intensively, over short periods. Typically, this has been done in an artificial context. This is in line with the above definitions which standardise the user population, environment, tasks and user support. This type of study can provide many valuable insights.

However, it suffers considerable limitations where our goal is to understand the way that users in their natural setting are able to use the editor. It cannot deal with issues that require longer periods of user activity and larger user populations. For example, a study may be required to observe people practicing highly infrequent, but possibly critical, commands. Also, what is the relationship between the observations of the many short duration studies? A series of long term studies that identify the major periods of development in the use of a text editor serves to define the context of the short term studies. Similarly, we need long term studies to learn about the development of a user’s skills and knowledge of an editing tool. Indeed, not long before the start of the present research, it was reported in a survey [3] that

"Text-editing is a skill everyone with an average education and knowledge of the task domain can master. The effect of prior knowledge and quality of training was not studied extensively, and we do not know any effects of training on asymptotic performance. The data available show that performance approaches the asymptotic range after about 50 hours, but the long-term benefits of practice such as increased flexibility and reduction in errors have not been studied."

The economic advantage of natural setting studies is considerable for research into long term usability. Conversely, under controlled conditions the means to reward the users for substantial periods of time has to be established. Even if a relatively modest approach is taken, the users must return repeatedly to do prescribed artificial tasks. For a substantial user population, this would demand considerable time from users and researchers alike.

It is desirable to appraise usability and the way that it changes over time as users employ the editor in tasks that are meaningful to them and working in a manner that is natural for them. This constitutes a rather narrow test of usability since users in their natural setting will be focussed primarily on a higher level goal. So their editing actions will be dominated by the ones they can do automatically [10, 11].

There are many elements in assessing the usability of a tool [18]. Some approaches involve detailed analysis of user actions to achieve specified tasks and to compare the observations with model sets of actions. We have been more concerned with obtaining an indication of long term learnability and user’s attitude to it. We wanted to get a picture of the facilities of the editor that were used and those that were not, to assess whether we could conclude accurately from this which facilities users knew about. We were also concerned to see how these changed over a very long time. In terms of a GOMS [4] model, we focussed on the ‘operations’ that the users knew how to effect correctly. Our methodology stops before the ‘methods’ and ‘selections’. For many users, with near minimal knowledge, the selections available to them do not involve many different methods for tasks anyway - so our level of analysis goes far enough to be of utility.

In this paper, we describe our study of people learning to use the sam text editor for up to three years. We present a toolkit of methodologies, software and analysis tools for appraising what people used and how this was enhanced over the years.

```

+. W10CharArray.p
+. data2
+. result3

program W10CharArrays(input, output);
{Written by 1031876:Oliver Twist:Fri_night:in the cave

    This program read a line of text from a file, deletes all
    contiguous duplicates, reads two characters on a second line
    (same file), and substitutes all occurrences of the first
    of these characters with the second.

Input  : A file containing two lines of text -
        line 1: the original line of text
        line 2: two characters - the character to be changed,
              and the character it is to be changed to.
Output : The line of text after the substitution,
        the line of text after the removal of duplicates, and
        the original line of text.
}
const
    maxlength = 40;
type
    indextype = 0..maxlength;
    arraytype = array[indextype] of char;
var
    A, B, C : arraytype;
    lengthA, lengthB, lengthC : indextype;
    12
    a
    a
    aaaaaa aaaaaaaaaa
    removedup
    readln(col
    Substitut
    WriteArra
    WriteArra
    WriteArray(A, lengthA)
end.

```

cut
 paste
 snarf
 look
 <exch>

Figure 1: Example of a sam screen

2. METHODOLOGY

2.1. Goals of the study

To watch and understand the development of methods of interaction with the editor, by a large number of users, over a long period.

To balance the conflicting goals of low cost monitoring with the goal of adequacy of data collection. The data had to be adequate to establish what people were able to use. There also had to be a sufficiently rich basis to follow up many of the questions that would inevitably arise.

To develop an approach that could be repeated with different user populations and software tools. Therefore it was important that the various costs associated with data collection and the subsequent analysis should be quite modest. Since our study was to be long term, the causes of concern included the machine related costs of cpu-cycles and storage as well as the human resources for management of the project.

A fourth important goal was to be mindful of the users who acted as unpaid participants in this study. To recognise their rights to privacy, to access the information obtained, and to have ultimate control over data collection.

2.2. The editor and computing environment

The `sam` editor was developed by Rob Pike at Bell Labs [21] for use by sophisticated system programmers, who needed to deal with large systems across multiple files. `sam` provides very powerful editing facilities and access to the Unix shell within the context of a fast windowing system. It is not in widespread use although there is a community of users. It is now the default editor for Plan 9 [22].

One essential element in the choice of `sam` was that it is fairly typical of the current and coming classes of text editors, where windows display files and many editing actions are performed with the mouse. The other critical aspect of `sam` is that it is fairly straightforward to learn how to do simple things in simple ways, yet it still has extremely powerful facilities available. This makes it rather like many other computing tools and so the trends in users learning its facilities are likely to apply elsewhere.

The `sam` display fits in an X window, within which special sub-windows are created and managed. In the example in Figure 1 the largest window contains a file of Pascal source text, other small windows on top of this show data. The "sam window", at the top of the display, is used for entering a powerful set of commands similar to `ed`'s. At the left hand side of each window there is scroll bar. The first button of the three-button mouse is used for window selection and pointing. Button two has a menu for cut and paste operations (see Figure 1), and button three's menu addresses file and window management.

The computing environment was based around MIPS minicomputers running Unix and X11 on NCD19 terminals. The default window manager for our users is `mwm`.

The population studied was the undergraduate students in the Basser Department of Computer Science at the University of Sydney. The use of `sam` in the three levels of a full computer science major was monitored over three full academic years.

There are five groups of novice editor usage: the first year class in each of 1991, 1992 and 1993 as well as the second and third year classes in 1991 when the undergraduate use of `sam` began. Thus there is a total of 2273 students who used `sam` during one or more calendar years. That is, they started `sam` during the year - but some elected not to use it much and others transferred to other courses.

There are also three sets of two-year use: the first two years for students starting in 1991 and 1992 as well as the data for the second year students who began their use of `sam` in 1991. We have 880 of these in total.

Of course, it also gives one set of three-year data, that for the cohort that began their first year in 1991 and did their third year in 1993. There are 316 of these. Not all 1991 entrants were in the CS3 class in 1993 - some repeated earlier years.

Over the period of study, the courses for first and second year students were fairly stable. All students were required to do similar tasks, mainly writing or modifying programs. This means that the users were doing the same tasks at the comparable period in each year. So one can, for example, compare the behaviour of users in the first year class over each of the three years of the study. (The students in the CS3 course had a large range of study options and were doing a variety of tasks.)

These data are summarised in the following Table. The unusually high number of starters in 1991 (1163) is due to the addition of the second and third year students in 1991 who had access to sam for the first time.

Years of opportunity for sam use	Year started in sam			Total Participants
	1991	1992	1993	
1 or more	1163	595	515	2273
2 or more	558	322	-	
3	316	-	-	

Table 1: Number of students monitored each year

2.3. Support for learning sam

The first year students received some instruction on the use of sam in their scheduled classes along with a brief introduction in their workshop notes and a reference manual. There would have been some variation between workshop groups in the quality and quantity of their tutorial instruction.

In 1991 - the year sam was introduced - the second and third year students received only the notes in the first year of use of sam. Importantly, the third year students had previously been using vi. This is significant as sam is from the same ed family as vi: the regular expressions and Unix interface ideas are common. So these students began their sam use with some pre-knowledge.

3. THE MONITORING TOOLS

3.1. The Design Goals

The overall goals we described in the last section define a number of subsidiary constraints on the design:

- **reliability, quality and utility** of data:
 - data to be useful for the goals of the study
 - as much flexibility as possible in terms of analysis programs
 - data to be stored for subsequent analysis
 - data collection to be continuous with low, or zero, loss
- **low cost:**
 - no significant system performance degradation during the day
 - low cost in storage
 - minimal interference with sam source code
 - low systems programmer cost
 - the system had to be highly reliable
 - the system had to run autonomously as far as possible
- concern for users' rights of **access, control and privacy:**
 - user access to the raw data
 - user access to summative data about them
 - intelligibility of the information kept about the user
 - avoid storing sensitive information
 - avoid storing information that is closely identified with an individual

3.1.1. Quality of data

As indicated in the above list, we needed to define the data that would be sufficient to establish which aspects of the editor the students were using. Moreover, we had to do this reliably over a very long period.

With respect to the technical aspects of data generation and management, we have confidence that there are few errors. The code modifications to `sam` were very carefully tested during their implementation, and were designed to minimise changes anyway. There can be slight problems at the margin if there is a system crash, but this should only affect a few commands in thousands per user. (Actually the monitoring output helped us find an obscure bug in `sam` during the first teaching week in 1991.) Also several concurrent invocations of `sam` cannot be handled perfectly, as it is not possible to merge all commands into a single sequence, even if that were desirable, because the time resolution is 5 minutes in the worst case [6]. This was not considered to be a serious problem as most of our users will not invoke `sam` concurrently anyway. Nor is it important for many of the analyses that we perform.

The most serious systematic source of error will occur if people use `sam` under other account names. This drawback is not a consequence of our particular approach to monitoring. Rather it is a function of this study being of users in their natural setting. There are three types of this source of error. First is the error in analysing a given student's activity during a period when another user, perhaps a laboratory demonstrator, does a few quick edits to help out. Another, more serious case, occurs when people "borrow" each others account, although we believe this is a small factor. These errors are likely to form a small percentage of the overall data set for an individual - perhaps a total of an hour of use out of a data set representing 20 to 250 hours. It will only affect individual profiles slightly, but should not alter system-wide aggregates. The third source of error is that some users may actually obtain considerable experience in `sam` via a group account, as happens for example in software engineering group projects in the third year. This makes it infeasible to determine a complete profile of activity for some individuals.

3.1.2. Low cost

Given our goals in terms of the duration and magnitude of the study, we had to be extremely aware of the costs at all levels. Our belief was that we could devise a monitoring mechanism that would be cheap enough to use in the long term. We believed that the amount of data could compensate in part for its quality.

The storage cost has been minimised by nightly collection, processing and compression of data log files. We have been much less concerned about the long term archiving of data which can always be done on tape if necessary. The day time cost is the extra disk space allocation for each student. With around 1000 user accounts active each year, we were restricted to storing only a few Kbytes per user each day. Even collecting 10Kbytes per day would require 10Mbytes extra disk - we actually need less in practice. Each student had a disk quota of around 3Mbytes, so our burden was small on a day to day basis. In fact, over a full year, we obtained a total of 70Mbytes of compressed data at negligible cost in terms of system performance.

3.1.3. Concern for users rights of access, control and privacy

Earlier work had pioneered the concept of user-viewable models [7] and we wanted to use these to give users access to the data collected. We also maintained this philosophy of accessibility in the way that we managed the raw data. All students were told about the monitoring and corresponding files held in their home directories, and that they could examine and, by implication, delete their own data. This gave the users final control - they could remove or alter monitoring information. Very few of the students appeared to delete or alter these files. Users can also set a switch that disables monitoring. In practice, this was only used by systems programmers who tended to have `sam` sessions which ran over many days and which interfered with daily collection and removal of monitoring files.

There are several aspects to the privacy issue. The first is that if all keystrokes are stored it is possible to reconstruct the text of files. There was no desire to do this even though it meant that we would lose a possible means of deducing intentionality about sequences of mouse and keyboard actions. The second issue concerns information that can be gleaned about the sorts of things a person is doing and which files were edited. We did not undertake anything with this data except research into user modelling. The third aspect is the access to the data by individuals outside the research project - perhaps for the purposes of assessment or comparisons. We have not released our data to anyone. Eventually, we may make our data publicly available, but with means of identification removed or obscured. Finally there is the question of anonymity of our student participants in any publications. Again, this is achieved through conformance with proper research ethics and standards.

3.2. The monitoring language

The monitoring language has been designed for near optimal collection and storage. It is intended for subsequent analysis by programs rather than for easy reading by people and is therefore rather terse. There are certain actions, such as the depression of a mouse button or typing text into a window, which generate a report in the monitor file for the user. Table 2 lists the different types of report. Each report has a unique identifier, for example *Me* for the *cut* menu item or *Ku* for the *undo* keyboard command, and sometimes some additional data as shown in the right hand column of the table.

Each monitor file has certain information at its head and tail. On invocation of *sam* the user identifier and a timestamp in seconds are reported, as shown by the first row of the Table. The "start monitor file" is reported (second row of the Table) either just before or just after this, depending upon the order processes get executed. This serves two purposes. First it tells us which version of the software wrote the file (we have been on version three since 1992), and second it provides a place for one line of free format comment should we wish to edit one into the file. This was included so that we could annotate problems or changes in the monitoring process if they were discovered during the monitoring. (As these problems did not arise we did not need to use this.)

The next two rows in the Table deal with the termination of a *sam* session. When *sam* exits, even in most abnormal circumstances, it does certain housekeeping work before closing down; the corresponding "shut down" report includes a timestamp and the total number of mouse commands and keyboard depressions during the session. The totals, timestamps and consistency checks can help us to determine whether any data has been lost, and if so, approximately when and how much. Upon normal exit by the user an additional report, "Exit", gives the total session time and various consistency check data.

Whenever a disk file is read, created or updated a File I/O report logs the filename, time and the byte count. The file size gives an indication of whether the display would need to be scrolled. The filename helps to identify the task because, for example, we know the names of various files that had to be edited during laboratory sessions. It is also possible to guess, from the filename extension, where the task is program editing or text editing.

Every typing run is recorded. The number of characters typed, the number of backspaces used and the time spent on typing are output. A typing run starts when the first character is input into a window and finishes when another report is generated, such as a mouse click. This data is useful in a number of analyses. For example, some of our work has needed to have an estimate of the user's typing speed, using the distribution of times to discount cases when the users may have stopped typing to talk, read notes or whatever [6]. This is because the best way to do things may differ according to how fast the user can type and because typing speed also gives an indication of the user's facility with computers. We were able to make a very good estimate of typing speeds by analysing this. The same monitor information is useful in assessing whether a user is making effective use of some commands. In particular, there are mouse commands that are easy to invoke by accident and this data helps establish the context so we can better assess whether a command use was likely to be intentional or not.

Report	Data Output
Invoke sam	userid, timestamp
Start monitor file	version number, optional comment
Exit sam	session length in seconds, consistency checks
sam shut down	timestamp total mouse cmds since last timestamp total kbd depressions since session started
File I/O	bytes read or written, filename, timestamp
Typing into a file	chars typed this typing run backspaces typed this typing run length of typing run in seconds
5 minute timestamp	timestamp total mouse cmds since last timestamp total kbd depressions since session started
Mouse click	none
Keyboard Command	none
Regular expression	none
Address	none
Error or Warning	none

Table 2: The major monitoring reports

At the end of each five minute period a timestamp is triggered by some other report, such as a mouse click. This makes it possible to keep track of time fairly precisely over an hour long session and with considerable accuracy over a year. Timestamps also facilitate the sorting and merging of data collected at various places during the day. Each timestamp takes the standard Unix form of the elapsed time in seconds since 1 January 1970. In addition mouse and keyboard depression totals so far for this session are output for possible consistency checking purposes. Even with this data redundancy, we cannot detect deletion of the whole file.

Each mouse click is reported without any additional output (see the eighth row in the left column of Table 2). For menu items this is straightforward, *Ma* for *select window* for example. When positioning in a file each click generates a report, so a double click actually writes a *Md* for *position cursor* followed by a *Mb* for *double click* - these can be cleaned up by postprocessing.

Erroneous actions generate reports. Errors and warnings (last row of the Table) produced by the editor, as well as internal errors that are identified by sam, are valuable sources of information for assessing usability.

sam has a powerful set of commands available in the command window through the keyboard, each generating a distinct report, eg *Kp* for *print*. Some commands trigger an additional report, such as *Kw* for *write file* triggering an I/O report. Some keyboard commands incorporate address operators, e.g. *A\$* for *end of file*, and regular expression operators, e.g. *R** for *zero or more*. Each and every invocation of these operators and commands is logged, so a single global substitution can generate a multitude of reports. It is thus possible to build up a detailed time-dependent experience profile of addresses and regular expressions.

The Appendix gives the main commands, addresses and regular expression operators together with their frequency ranking. Figure 2 shows an example of a monitor file.

H ronny 687597236	<i>Start session</i>
%3:	<i>This is version 3 of the monitor file format</i>
MqMyEa	<i>Open a window for a non-existent file (get error)</i>
I11 0 4	<i>Insert 11 characters in 4 seconds</i>
MaKqFnKq	<i>Select command window, quit</i>
S 687597248 3 15	<i>Closing time stamp</i>
D12 2 2	<i>Quit marker (after 12 seconds)</i>

Figure 2: Example of part of a monitor file

3.3. The monitoring software

The monitoring data is generated by statements written into the `sam` source code. These are positioned in the most expedient places in order to keep modifications to a minimum (see design goals above). An option in the `sam` Makefile enables or disables compilation of these statements. On the whole this is quite satisfactory, although there are idiosyncrasies; for example, commands with addresses and regular expressions are reported in an order which is counter-intuitive; and certain postprocessing is necessary to correct misleading reports such as *position mouse* followed by *double click* for double click. It is just more convenient to trap this sort of thing later.

`sam` actually consists of two processes, the host and the terminal - a design decision that took advantage of the Blit environment in which it was developed. There is a synchronisation mechanism between these two processes which is created shortly after the initial `sam` process forks off what becomes the host process. Once `sam` is fully invoked, each process writes out to a file called `$HOME/.sapid`, where *pid* is the terminal process' id. (It is possible to invoke `sam`'s host process alone, for use as a command based editor on an ordinary terminal, but this has been disabled for our students.) The locking mechanism ensures that there are no conflicts so long as each monitor command is output as it is generated and not buffered. Calculations showed that this style of output would not become a significant factor in total system I/O. Theoretically there can be slight synchronisation difficulties during startup, but this has not been observed over the three years of operation. The design goals we articulated above encouraged this type of solution. We minimised the problems by disabling access without the window display, and then adopted a source code modification which was easy but theoretically might fail occasionally at startup. Operationally, this has been justified by experience.

We did consider the possibility of writing the data straight to a "secret and safe" location during editor sessions. This would have had several drawbacks. First, loss of freedom of access by students to the data on the day of generation. Second, a possibility that the supporting daemons would fail. Third, it would require special system privileges. Fourth, more complex `sam` source code modifications would be required. Our simpler approach works extremely robustly and has provided further evidence that the design goals were sensible.

3.4. Data Management

During the day, files are created for each `sam` session over a variety of machines by hundreds of users. The machines themselves were grouped into clusters for first year undergraduates, second and third years, honours, postgraduates and staff. A shell script is run nightly to remove all monitor files from home directories and append them onto the data previously obtained for each user. During collection small amounts of postprocessing occurs, for example to tidy up double clicks or encrypt usernames (now discontinued), and data is compressed with GNU `gzip`. At this stage each session is given an independent date stamp; this has proved invaluable when dealing with ad hoc problems or analyses. At the end of each academic year, a large rationalisation is performed to merge files that may exist on different machine groups and as far as

possible to ensure that all sessions are stored in date/time order. The end result is one monitor file per user per calendar year. For 1991-3 we have 4063 such files.

There are several advantages to this approach: the monitoring system requires no special permissions to be granted for the `sam` process to operate and it is not dependent upon daemons or other systems being available. The only system privileged part of the monitoring is in the collection phase, which is run automatically each night. Corrupted data is localised to a single `sam` session. Although there is a tradeoff between this file collection and other methods, we are very satisfied with the present arrangements which require little attention except at the end of each year.

4. ANALYSIS

Our main approach to the analysis of the monitor data has been to construct a collection of tools, each of which analyses one simple aspect. These can also be combined to build more complex analysis tools. This section gives the flavour of the types of analysis tools we have constructed.

The raw data is not user-friendly. In practice we almost always postprocess the data through a filter, `sscan`, which can optionally produce explanatory English text strings and always places one command per line. The design of the monitor data has made it straightforward to build a wide range of small tools that each determine an interesting element of the ways that `sam` was used. For example, `elapsed` lists each command invoked in a given input file together with its total use and time of first use. Similarly `cmds` builds a table of weekly use for a selection of commands from a given start time. Graphing this gives a powerful overview of the changes in use of a command.

There are about 90 actions a user could try out and thereby generate monitor data. As has been observed in other studies of text editor usability, most users did not try out more than half of all the possibilities.

4.1. Some tools to describe long term change

4.1.1. Learning by exploration

Although our students had some guidance in the initial use of `sam`, the main mode of learning in the longer term was to try things out on an individual, exploratory basis. Of course this is common for many, many computer users. Recently, researchers [20, 9, 24] have started to look at this phenomenon in its own right. The approach adopted here is closest to that of Jordan et al [14].

The First Two Hours

It is not transparently obvious how to use `sam`, but the first attempts are very pedestrian. Figure 3 shows the time it took for the first intake, in 1991, to try out the 10 most commonly used commands (of the first semester) during their first two hours of `sam` practice. In this instance, a trial corresponds to an invocation of a command, whether deliberate or not and regardless of success. (It could not be more generous.) The x-axis shows the time and the y-axis the percentage of users who had tried 6, 8 or all of the ten most common commands at time x . It can be seen that 50% of the class got to six commands within a few minutes, but 8 commands took much longer, and 10 was barely reached in two hours by the majority. We realised that it was important for the guidance given to the students to be as focused as possible. Accordingly, in the following year we changed what was taught in these first two hours, and although the class was much smoother and enjoyable for the students, the profile did not alter much. We conclude that people need time - hours - as well as guidance to settle in. Of course, just because the rate of exploration may not have altered between the two years, we cannot conclude that the quality of the learning experience was the same.

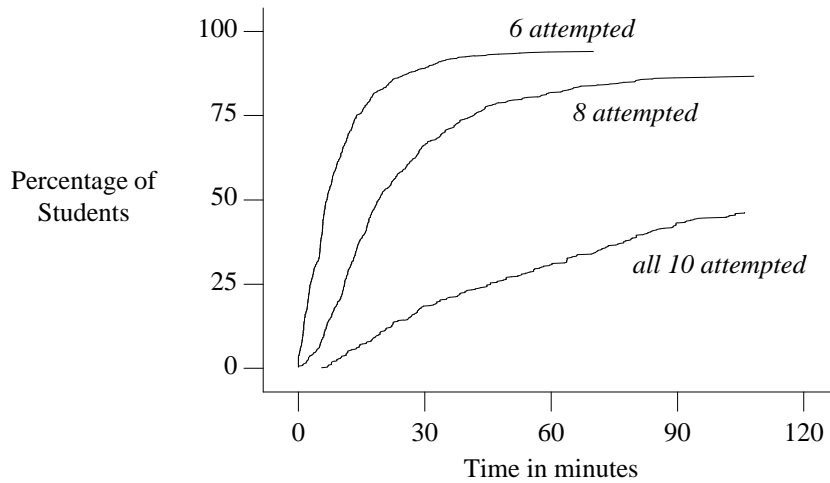


Figure 3: Time Taken to Attempt the 10 Most Common Commands

Figure 3 was calculated from our raw data by post-processing the output of the `elapsed` tool to filter out all unwanted explorations. Then a series of sorts and merges eventually gave the required percentages, which were used to generate the graph.

Long term slow exploration

Where the above discussion dealt with just the ten most commonly used commands, we now consider the rate of exploration of the full command set. In fact users tried out about 25 commands in the first couple of hours, but most of these were spurious mouse actions. We call one of these trials an *exploration*. It is defined as occurring the first time a user invokes a new command for whatever reason and regardless of outcome. It may represent the first step on the path towards learning a command, and the time it occurs provides a lower bound on when that command could be learnt. The total explorations up to some point in time give an upper bound on the number of commands that could have been learnt by then.

It turns out that users go on exploring new commands over the years. This is evidence that the inventory of commands for each user may be slowly but steadily increasing. Figure 4 shows a histogram of total explorations by a random sample of 63 users who entered the university and started computer science in 1991 and who used `sam` for three academic years until October 1993. Some of these students may be repeating years if exams were failed. There is a mode developing at around 41 explorations - about 15 more than after 3 months of `sam` use. Figure 5 shows the pattern over the full three academic years for an individual, student A. We shall see later that this user goes on to do something extraordinary. Often an exploration is not followed up by much use of the particular command, but we regard this as part of the challenge in our analysis.

To produce the histogram, each user monitor file is processed by `elapsed` and `wc` giving the number of explorations per user. It is then a simple matter to draw the histogram. The individual exploration curves are generated by `elapsed`, but care needs to be taken to turn the times into meaningful units, eg week numbers.

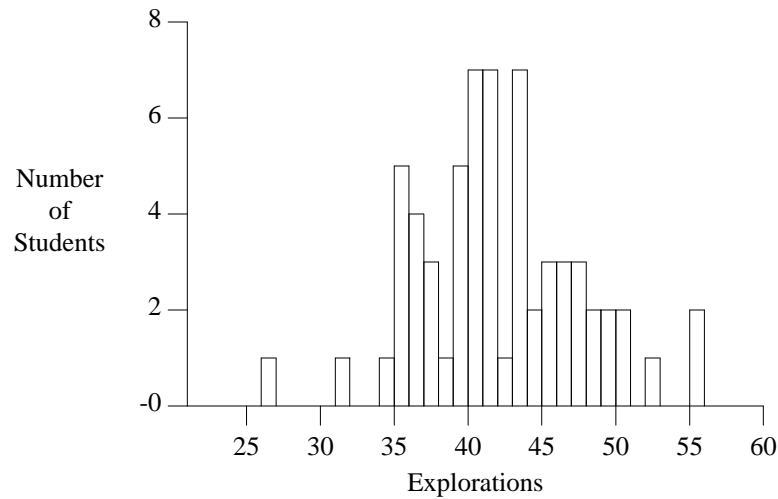


Figure 4: Histogram of Total Explorations During 1991 to 1993

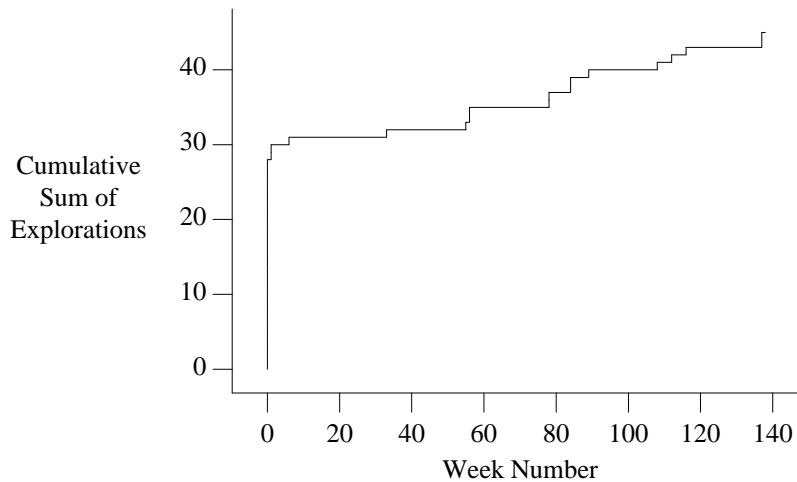


Figure 5: Explorations by Student A

4.1.2. Drop-out Rates

One of the advantages of natural studies is that users can be observed to decide not to use a system. As Eason [10] points out

"... one of the most telling indicators of non-usability is when a user stops using a system. How do we arrange for our subjects to visit our laboratories in order not to use our systems?"

The laboratory in our case was the cohort of CS3 students who had the option to start using sam in 1991. This class was told of the existence of sam and was given the set of notes prepared for the CS1 class, but students were left to decide for themselves whether to try it out. Of the 213 people, 75 never tried it. Of those that did, many gave up during the year. Figure 6 shows the distribution of dates when sam was used for the last time by each student.

Some of this class were interviewed to try to find out why they gave up. A large number found the first few minutes very off putting as they probably did not achieve much - like the CS1 class. Others were prepared to give sam a try, but most still found it less comfortable than vi. Significantly, some of this year *were*

prepared to learn the xedit editor on a voluntary and unprompted basis. Some of the class saw no benefit in trying sam and many more failed to profit from their initial experiences. The histogram of explorations for these students differs from that for CS1 in that there are a number of students who explored very little. It took the students a lot of time and effort to guess what to do in order to perform useful tasks - in Jordon et al's terms [14], sam was not very guessable.

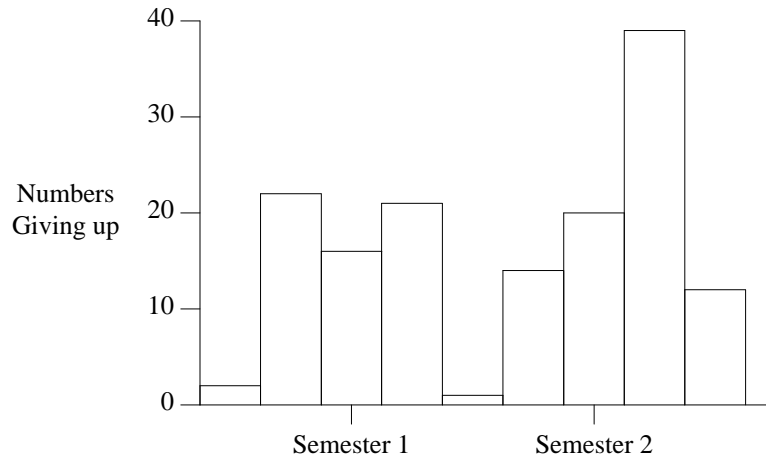


Figure 6: Histogram of CS3 Students Giving up During 1991

Before we leave this, we point out that the interpretation of such data needs considerable knowledge of the context of the study. That sam is not widely available is both a curse and a blessing.

Consider first the advantages. Because almost all our users could only use sam on our machines, our monitoring gives a complete record of their use of it (except for the limitations mentioned in section 3.1.1).

Now, the problems with sam's limited availability strongly affect the dropout rate, especially for these senior students who spent at least half their course load on Computer Science. Many did a good deal of their programming and course work outside the department, on machines that did not have sam. Frequently, those machines did have vi and these students had less reason to invest the effort of adjusting to the new editor, especially when it has a very different feel from vi. Even more than this short term issue, the students generally perceived sam skill as being unlikely to be useful after they completed the course (and for most, CS3 was in their final year).

There are several other factors. There was some kudos associated with using vi (over the simple looking sam). The very students most likely to explore a new tool were most likely to be influenced by this. In addition, these students often gained extra machine access by using ascii terminals where they were unable to use sam.

Another factor is sam's very mouse-intensive style. This posed two problems for the CS3 students. First, they were mainly accustomed to vi which is heavily keyboard based (though the X-interface makes it possible to do some pasting of text). By CS3, the students are quite effective users of the keyboard. A related detraction from sam is due to the poor state of some of the mice on the X-terminals used by these students.

It would seem that for the CS3 class, sam did not appear to offer enough advantages to invest the considerable effort needed for students to be able to use it automatically. For those who felt that they knew vi very well, the available information about sam and their first experiences did not show the potential power of sam as a programmer's editor.

The calculation of drop out rates is very easy. The output from `sscan` is processed by `grep` to obtain "shut down" reports. The last one in each user file is the drop-out time. These can then be sorted to give the graph. The most difficult thing is deciding which files belong to genuine CS3 students - a student record issue, not a monitoring one.

4.1.3. Long term sophisticated shifts

A major concern in our work has been to study long term changes in use of the editor. To illustrate the approach we take and the outcomes that can be derived from our methodology, we now discuss one class of operations in some detail: ways to write out a file.

There are three principal methods to write out files. The steps in using these are shown in Table 3. To make the analysis more meaningful, we need to explain the ways these work.

Step	Command	Button-3 write	Walk-through menu
1	select command window	click button-3	click button-3
2	type 'w' [optional filename]<return>	select 'write' from menu	select file name
3	optionally reselect editing window	click on window to save	slide across
4			select 'write' from menu

Table 3: Steps required for the three ways to write a file

Using the command window to write out a file might appear to be the simplest method. In practice, this is not so. Indeed, a GOMS or Keystroke Level analysis is based upon models for expert use of the editor and this makes that type of analysis misleading for our situation as we describe below. In addition, the command-write method is the only one in Table 3 that requires use of both mouse and keyboard and the switch to and from use of the keyboard is known to be slow (as for example, observed in [29]).

The first step, selecting the command window, ought to simply require a button 1 click in the command window. For an expert, who has ensured that the command window is always visible, this is a very simple step. However, for our students, there are several potential difficulties. Firstly, they often manage to accidentally obscure the command window: then selecting requires using the button-3 menu and selecting `~sam~`. We know that most of our users never used this facility so an obscured command window meant that they did not know how to get the command window.

A further difficulty is due to the fact that one button-1 click on a window selects a window and a second moves the cursor in it. For the case of the command window, this would mean that subsequent typing of 'w' would not effect a write. For a user who did not realise they had clicked twice, or who did not realise what effect the second click had, or who could not move the cursor to the end of the command window, this caused an impasse. In practice, many of our users had extreme difficulty with the command window because of these problems.

In our first year of monitoring, students were taught the command write first because it is the most general: if a new file buffer is created, it is the only way to associate a name with it and write it out. Moreover, any write can cause error reports and as these appear in the command window, we wanted students to look at that on each write.

The second means of writing requires, as shown in the table, selecting the button-3 menu and from this the write command, then clicking on the window to write the file.

The third method, which is not part of the basic `sam`, was added by our programmers because they felt it was useful. It is also based on the button-3 pop up menu but requires selection of the file name, sliding across from this to make a submenu appear and then selecting the write command from this. This might

seem more complex than the second method. However, where the user has just one file, its name always appears at the very bottom of the button-3 menu and, if the user continues to use the write-option in the slide-across popup menu, it is automatically selected by a simple slide action.

It is when the user has many files that the second method is more convenient than the third. The write command is in the middle of the button 3 menu and is harder to find than the filename where there are only one or two files. However, where there are many files (as in typical C programs), the user will usually need to scan through them to the one they want to write. Moreover, in that case, many of the files will have similar names and this scanning requires considerable care (even though they are sorted in the menu): it is easier, in this situation, to establish the file to write by simply clicking on its window, as required for method 2.

In summary, the command-write is essential for writing unnamed buffers. It has only three steps for expert users but requires swapping between mouse and keyboard and back and it is conceptually far more difficult for our user population. By contrast, the other methods are easier. The context defines which is faster and simpler to use. Interestingly, most students actually found the third - the walk through menu method - of their own accord and stuck with it almost exclusively.

Figure 7 shows this pattern for student *B* - actually one of the top explorers - and it is worth noting how much swapping around between the methods occurred in the first few weeks before the style became set.

B did go on developing, for example, in week 127 a new pattern of sustained use of the *undo* command developed.

Student *A* was different. In terms of preferred file write method, this user behaved much like the others for the first 80 weeks. Then there was increased use of the alternative methods, and finally the student "flipped" and almost completely gave up the walk through method in favour of click and point. This is graphed in Figure 8. The explanation for this lies in the number of active filenames that were used in each sam session, which rose towards the very end of the three-year period.

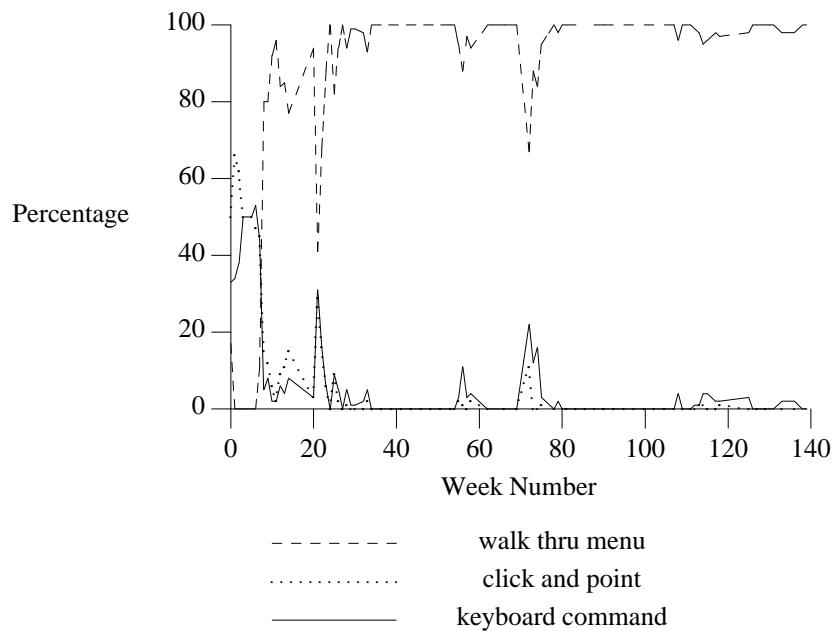


Figure 7: Use of the Three File Write Methods by Student B

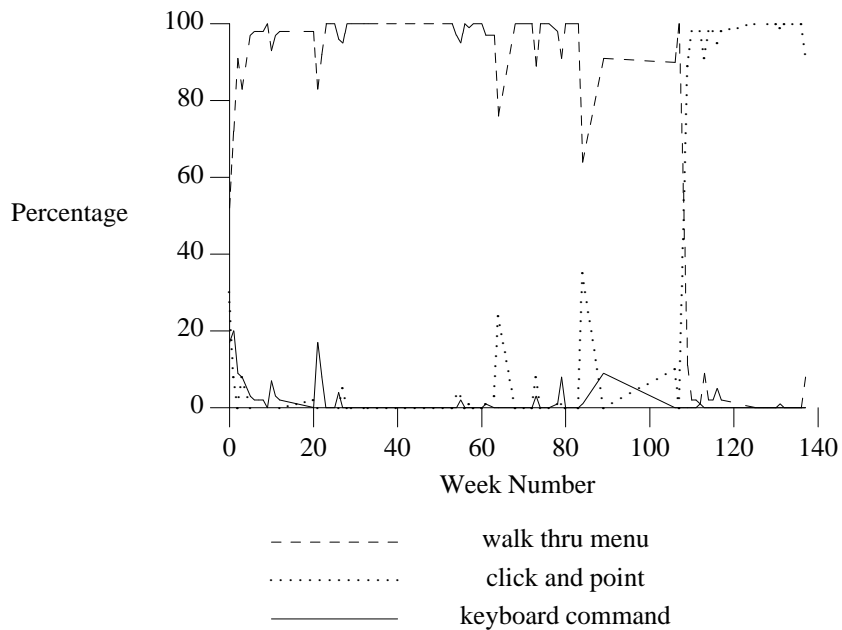


Figure 8: Use of the Three File Write Methods by Student A

4.1.4. The z-curve constraint

It seems plausible that the relative frequencies of commands would be useful for tracking changes. We are looking into this in some depth but for the present point out that in sam, at least as used by our sample of 63 three-year users, there is a statistical relationship between the relative frequencies which can be seen by eye. Figure 9 is a plot of the command rank along the x-axis and that command's relative frequency on the y-axis - drawn to log scale. It can be seen that it is almost a straight line at one level of detail. This informally indicates that the command frequencies are exponentially distributed. We have found this to be *approximately* the case for individual users and over smaller time frames as well. It has been found that Unix command frequencies conform to Zipf's Law in some cases [13,12]. Our distribution does not conform to this law, but the idea is similar, hence we call it the *z-curve*.

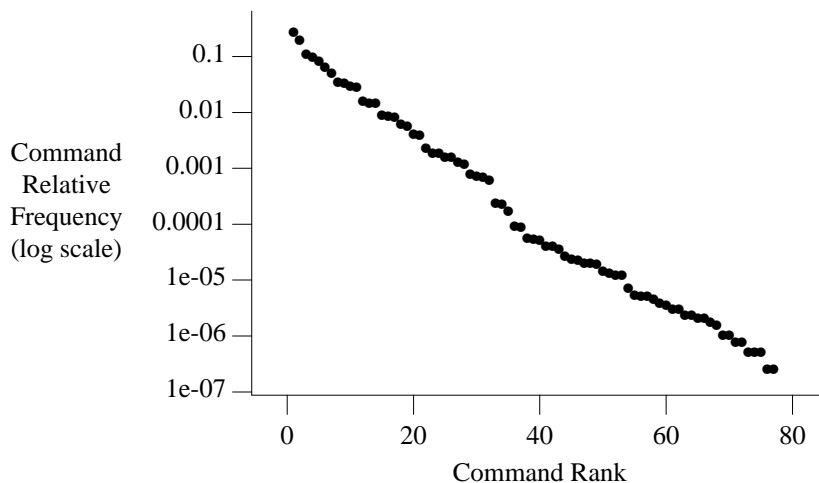


Figure 9: Z-curve of Command Relative Frequencies During 1991 to 1993

In order to understand the importance of this, consider our student A. File writes comprise about 5-10% of A's total usage. So, if a command suddenly jumps from nowhere to 7% of the total, its rank alters and so does that of the one it displaced. But in the long run the z-curve, as shown in Figure 9, appears to hold, so this says that if some rank goes up, others will have to come down. We thus have a measure of resistance to change. The *z-curve* becomes a *constraint*.

The creation of a z-curve graph is very easy. The entire monitor data is processed in one stream by elapsed giving a command count. Then it is merely a matter for sorting, calculating ranks and logs via a one line awk program, and finally graphing the data.

4.2. Individual usability assessment

We have developed tools that analyse the monitor data for an individual and build a user model in the um representation [16, 17]. This user model is actually a summary of information about the user. In this context, the information reflects what the user appears to know about sam from the way that they have used it.

We can make the information in this structure available to users both directly and via the use of the viewer program. An interesting outcome of this is the notion of individual usability where we can present the extent to which a user has learnt the facilities of the editor.

Figure 10 gives an example of a display of an individual user model produced by our viewer [7, 8]. This shows the elements that the user is judged to know about as white circles and those that they do not know about as dark circles. Where the system was unable to determine whether the user knew an aspect or not, it displays concentric circles. Shapes other than circles are used to display modelled aspects other than the user's knowledge (and these are discussed in [7]).

This means that the user model display provides the user with a clear and concise summary of the usability of sam for them as an individual. We have found that this opens a number of interesting opportunities.

Firstly, we used the viewer in interviews, discussed in [6] to validate our conclusions about the aspects the users really felt that they knew. The students were asked to explore the models and assess their accuracy, noting any errors. They were also asked to explain what it meant to 'know' an aspect. The most common response was essentially a functional assessment: they knew something if they used it confidently.

In general, they agreed that the models were a good reflection of what they knew. This gives us confidence in the effectiveness of the tools for analysing the monitor data [6]. Most of these tools are simple: they count the frequency of use of a facility and once it passes a threshold, the user is deemed to know how to use the command.

We have also explored the use of somewhat more sophisticated analysis to identify accidental mouse commands. We used this for a command which is relatively sophisticated (*xerox*) and gives two windows in one file. It is only useful when the file is too big to fit in a single window. It is extremely useful in programming where, for example, one is adding some code and needs to also add declarations earlier in the same file. Because *xerox* is selected from a menu, it is easy to select it accidentally [9]. Then, when the user fails to sweep out a window for the new view of the file, it is most likely that the user will not even be aware that they made that selection.

Not surprisingly, we observed a quite high number of invocations, especially in the first hours of sam use. To separate accidental menu selections, we used four pieces of context to help establish proper invocations. We ignored all uses in the first hours of sam use. We only counted invocations if the file involved was not tiny, if they were followed by the sweeping out of a window, and if there were some actions on the file after

this. In addition, we required a threshold number of uses of this (and each other) command so that the model is not affected by limited sam use by another user on this user's account (as, for example, where a tutor cleans up some problem). This gave quite satisfactory assessment of user knowledge for this type of attribute.

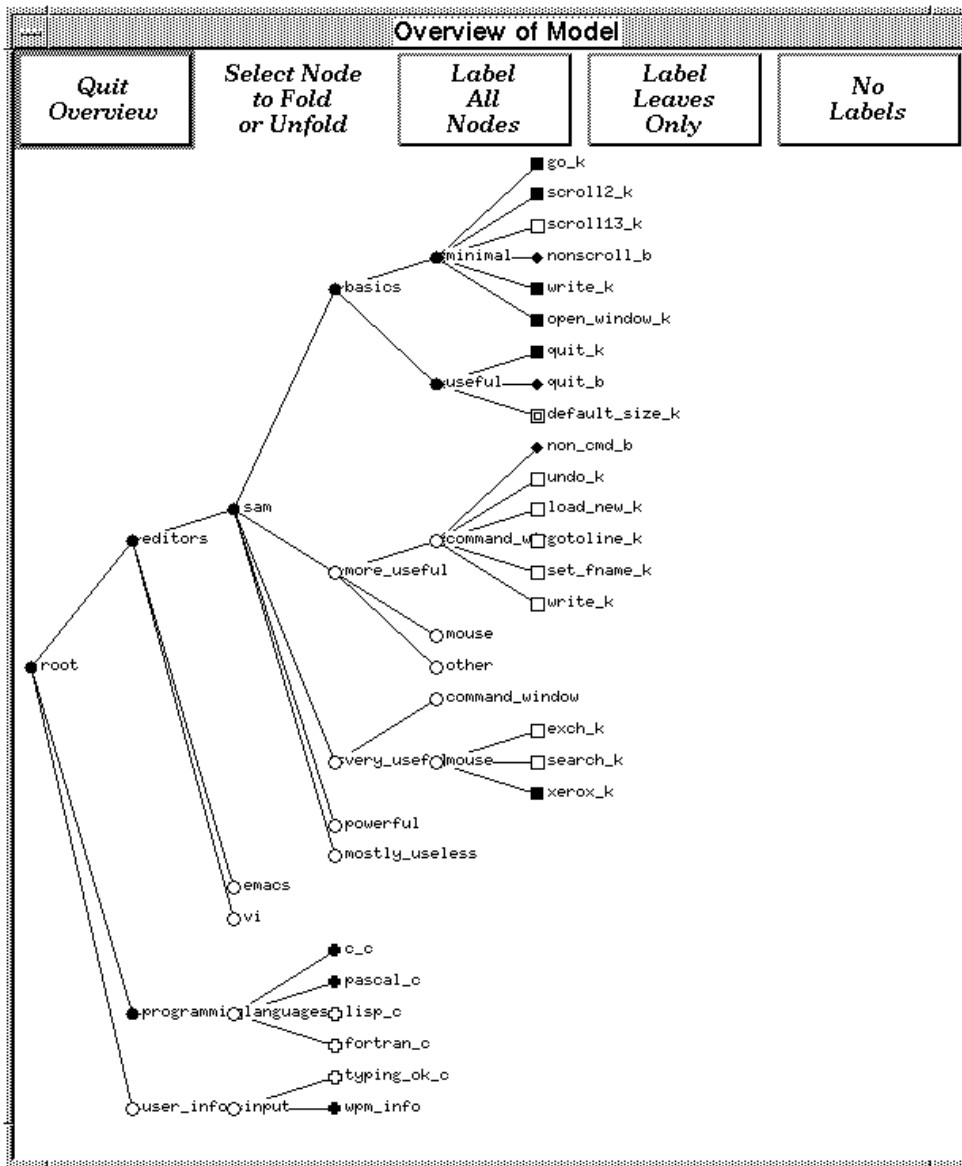


Figure 10: Example of display from a user model viewer

The process of modelling the user's knowledge and making that accessible has some interesting side effects. The viewer can explain any of the elements that it displays, and it can also provide explanations for the way that it concluded whether or not a user knew an aspect. Some of our users have explored these

explanations quite thoroughly. (We monitor use of the viewer.) In some cases, users have learnt more about sam from the viewer, often employing frontier learning, where they explore the aspects that are shown as unknown but close to ones they do know. Also, some users have examined the explanations of some sam facilities and decided that they do not want to know about those.

Making the user models available to users may well have affected our study because it alters the learning environment. To deal with this difficulty, whilst still allowing us to explore the use of these models, we made them available only to a small part of the class. We can separate this group out in our analyses, as indeed we did, to assess the learning effect of such access to the user model.†

5. CONCLUSIONS

This paper has reported work aimed at the assessment of the long term usability of a text editor. We have recognised the important criteria of learnability and user's attitudes, and appraise them in terms of the aspects that people seem to have learnt at each period in time, their patterns of usage, and other measures.

We have argued that long term studies in the natural work context are important and have demonstrated that they can be undertaken at reasonable cost. We have discussed our toolkit including methodologies, software tools, analysis tools and measures.

We have articulated a set of design goals for a long-term data-collection system and shown that they can be met. We have paid particular attention to robustness, rights of user access, privacy and cost. These will provide a point of reference for further studies.

We have obtained a very large data base comprising actions of over 2000 students for up to three years. This data is continuous from the time of first use of the editor, and it therefore represents a unique collection of both transient and steady state behaviours.

It is important to note the portability of our method. Not only can it be easily adopted in any Unix environment where sam (and its source) is available, but the approach of making the monitoring stand-alone and separate from the collection and analysis phases makes this approach portable to the investigation of other editors. Indeed, it could be applied to other tools and systems.

One of the important tools we have developed, the user model viewer, uses this command data to generate individual models. The users studied agreed with the individual usability summary it gave. This means that we have provided a means of communication about usability issues between the user and the researcher.

Studies of Unix command frequencies have indicated an underlying conformity with Zipf's Law - the hyperbolic distribution - whereas similar analysis of our editor data has suggested an exponential distribution. Error rates may be similarly distributed. We believe such underlying distributions in command frequencies form basic constraints on the rate at which users can adapt. This is an exciting avenue for further study.

The nature of our study has enabled us to devise new measures, especially the exploration activity. We can also obtain command uptake profiles and substitution curves. We also have evidence of sam drop-out rates for one cohort of students - the third year students who could switch over to sam in 1991, having primarily used vi in their earlier undergraduate years. Taken together these present very clear evidence that the usability of sam was adequate for some of our users, but not for others.

† Note that we felt bound to avoid disadvantaging one group of the class. So, students in the sam control group (and getting no coaching on it) were given some advice in another domain.

We have added to the body of knowledge that the differences between individuals are more pronounced than suggested by the gross statistics of use across the whole population. So, for example, the percentage of students using the different file write commands gradually changes, but this masks very marked differences in use by an individual from week to week.

The tools that we have developed provide a framework for tackling the considerable difficulties of studying long term acquisition of great skill. A very large initial population is required in order to obtain even a few eventual power users. Very long time scales are essential since skill acquisition is so slow. It may be characterised by some sudden shifts in patterns of use between a few commands but is more a process of creeping change, or annealing, overall. Of course, this is very much what good "flexibility and attitude" will promote.

We believe that it is important to continue studies like this one where we focus on users in real situations. This complements the carefully contrived laboratory settings of many other studies. One of the unfortunate effects of natural-setting studies like ours is that one needs to be rather cautious in interpreting the results observed. This is why we felt compelled to undertake such a large scale and long term study and, in turn, this is why we regarded monitoring as the natural basis for data collection. (Indeed, we are continuing the monitoring.) However, we readily acknowledge that data from other sources, particularly interviews, are important to complement the picture.

Our observations are likely to be generalisable to other populations of users of sam. However, a single study, even with the considerable scale of ours, is not enough.

The primary goal of this paper is to share our methodology for assessing the long term usability of tools like the sam text editor. The methodology is low cost and permits the study of a large population of users over the long periods of their development as users of a tool. Our study should be the first in a series that develops a broader understanding of long term usability.

Acknowledgements

This work was partially funded by Telecom Research grant Y05/04/34. We are grateful to our other partners in the sam study, Kathryn Crawford and David Benyon. We are also indebted to the programmers at the Basser Department of Computer Science for their support and expertise in the creation of the tools for this project, its maintenance and overall support. Also, thanks to Marion Cottingham.

Finally, but importantly, we thank the thousands of users who kindly participated in this study by allowing us to monitor and study their use of the editor. We particularly thank the hundreds who we had occasion to deal with us directly in doing cognitive profile tests (to support our interpretation of the analysis) and those we interviewed.

References

1. Allwood, C M and M Eliasson, "Analogy and other sources of difficulty in novices very first text-editing," *Intl J of Man-Machine Studies* **27** pp. 1-22 (July, 1987).
2. Booth, P, *An introduction to human-computer interaction*, Lawrence Erlbaum Associates, Hillsdale, NJ (1989).
3. Bosser, T, "Learning in man-computer interaction," ESPRIT Research Report, Project 385, Springer-Verlag (1987).
4. Card, S K, J P Moran, and A Newell, *The psychology of human-computer interaction*, Lawrence Erlbaum, Hillsdale, NJ (1983).
5. Chapanis, A, "Evaluating usability," pp. 359-395 in *Human factors for informatics usability*, ed. B Shackel and S Richardson, Cambridge University Press, Cambridge, England (1991).
6. Cook, R, *Viewable individual user models for a text editor*, Honours Thesis, Basser Dept of Computer Science, University of Sydney (1991).
7. Cook, R and J Kay, "Tools for viewing um user models," SSRG Report 93/3/50.1, SSRG, Dept of Computer Science, University of Sydney, Australia (1993).
8. Cook, R and J Kay, "The justified user model: a viewable, explained user model," pp. 184-190 in *UM94: User Modeling Conference*, , Hyannis, Cape Cod, USA (1994).
9. Draper, S W and B Barton, "Learning by exploration, and affordance bugs," pp. 75-76 in *Adjunct Proceedings of INTERCHI'93*, ed. S Ashlund, K Mullet, A Henderson, E Hollnagel, and T White, ACM, Amsterdam, Netherlands (1993).
10. Eason, K D, "Towards the Experimental Study of Usability," *Behaviour and Information Technology* **3** pp. 133-143 (1984).
11. Egan, D E, "Individual Differences in Human-Computer Interaction," pp. 543-568 in *Handbook of Human-Computer Interaction*, ed. M Helander, North-Holland, Amsterdam (1988).
12. Greenberg, S and I H Witten, "Directing the User Interface: How People Use Command-based Computer Systems," pp. 349-355 in *Analysis, Design and Evaluation of Man-machine systems - Proceedings of the Third IFAC/IFIP/IEA/IFORS Conference, Oulu, Finland*, ed. J Ranta, Pergamon Press, Oxford, UK (1988).
13. Hanson, S J, R E Kraut, and J M Farber, "Interface design and multivariate analysis of UNIX command use," *ACM Transactions on Office Information Systems* **2**(1) pp. 42-57 (1984).
14. Jordan, P W, S W Draper, K K MacFarlane, and S McNulty, "Guessability, Learnability, and Experienced User Performance," pp. 237-245 in *HCI '91 People and Computers VI: Usability Now!*, ed. D Diaper and N Hammond, Cambridge University Press (1991).
15. Kay, Dana S and John B Black, "Knowledge transformations during the acquisition of computer expertise," pp. 268-303 in *Cognition, computing and cooperation*, ed. S P Robertson, W Zachary, and J B Black, Ablex, Norwood, New Jersey (1990).
16. Kay, J, "um: a user modelling toolkit," *Second Intl User Modelling Workshop*, p. 11 (1990).
17. Kay, J, *The um toolkit for reusable, long term user models*, SSRG Report 94/3/36.2 (March, 1994).
18. Lindgaard, G, *Usability testing*, Chapman and Hall (1993).
19. Mack, R L, C H Lewis, and J M Carroll, "Learning to use office systems: problems and prospects," *ACM Trans. on Office Information Systems* **1** pp. 254-271 (1983).
20. Mul, S de, H van Oostendorp, and T White, "Learning user interfaces by exploration," pp. 135-142 in *Human-computer interaction: from individuals to groups in work, leisure, and everyday life. Proc 7th European Conference on Cognitive Ergonomics*, ed. R Oppermann, S Bagnara, and D R Benyon, Gesellschaft fur Mathematik und Datenverarbeitung, Sankt-Augustin: GMD (1994).

21. Pike, Rob, "The Text Editor sam," *Software Practice and Experience* **17** pp. 813-845 (November 1987).
22. Pike, Rob, Dave Presotto, Ken Thompson, and Howard Trickey, "PLan 9 from Bell Labs," pp. 1-9 in *Proc of the summer 1990 UKUUG Conf*, , London (July 1990).
23. Poller, M F and S K Garter, "A Comparative Study of Moded and Modeless Text Editing by Experienced Editor Users," pp. 166-70 in *Human Factors in Computing Systems. CHI '83 Conference Proceedings*, ACM, New York (1983).
24. Rieman, J, "Learning strategies and exploratory behaviour of interactive computer users," PhD Dissertation, University of Colorado, Department of Computer Science (1994).
25. Roberts, T L and T P Moran, "The evaluation of text editors: methodology and empirical results," *CACM* **26**(4) pp. 265-283 (1983).
26. Rosson, M B, "Patterns of Experience in Text Editing," pp. 171-175 in *Human Factors in Computing Systems, Proceedings of CHI '83 Conference*, ed. A Janda, North-Holland (1984).
27. Sebrechts, M M, P L Marsh, and C T Furstenburg, "Integrative modelling: changes in mental models during learning," pp. 338-398 in *Cognition, computing and cooperation*, ed. S P Robertson, W Zachary, and J B Black, Ablex, Norwood, New Jersey (1990).
28. Shackel, B, "Usability - context, framework, definition, design and evaluation," pp. 21-38 in *Human factors for informatics usability*, ed. B Shackel and S Richardson, Cambridge University Press, Cambridge, England (1991).
29. Young, R M and A MacLean, "Choosing between methods: analysing the user's decision space in terms of schemas and linear models," pp. 139-144 in *Human Factors in computing systems, Proc CHI' 88*, ed. E Soloway, D Frye, and S B Sheppard, ACM (1988).

Appendix

MOUSE COMMANDS		
Rank	Button	Meaning
12	left	double click
5	left	drag
1	left	position
10	left	select window
8	middle	cut
26	middle	exchange
19	middle	look
46	middle	no scroll
11	middle	paste
53	middle	scroll
24	middle	search
39	middle	send
42	middle	sever
14	middle	snarf
30	right	close
28	right	close (walk thru menu)
25	right	new
32	right	reshape cmd
13	right	reshape default size
23	right	reshape sweep size
9	right	select file from menu
15	right	write
7	right	write (walk thru menu)
33	right	xerox
37	right	xerox (walk thru menu)
27	right	~~sam~~
4	left	scroll
3	middle	scroll
6	right	scroll

TYPING	
2	Insertion of Text

WINDOW COMMANDS		
Rank	Identifier	Meaning
41	C	Change directory
51	D	Delete file
60	=	Print address of range
50	n	Print menu of files
62	p	Print text range
16	q	Quit
44	f	Rename file
34	e	Replace & edit file
64		Replace (pipe)
76	c	Replace text
52	!	Run Unix command
61	g	Search, run command
73	x	Search, run command on matches
72	y	Search, run command between matches
68	v	Search, run command when not found
71	>	Send range to named Unix command
43	b	Set current file
22	B	Set current file, loading file(s)
17	L	Set dot to given address
70	k	Set position mark
40	s	Substitute
29	u	Undo
21	w	Write buffer to given file
36	r	Read in named Unix file

<i>Address Operators</i>		
63	+	2nd address eval at start of 1st
48	,	substring from 1st to 2nd address
58	-	2nd address eval back from 1st
59	;	substring from 1st to 2nd address
67	#	Character position
35	?	backward search
49	.	dot (current position)
54	\$	end of line
77	"	filename
31	/	forward search
18	l	line number
56	'	pre-set mark

<i>Regular Expressions</i>		
75	\$	End of line
45	(Open grouping
57)	Close grouping
55	*	Zero or more
69	+	One or more
47	.	Any char except newline
74	?	Zero or One
65	[Character class
20	\	Esc, newline or tag
66	^	Start of line