Int. J. Human–Computer Studies (1994) **41**, 385–397

Estimating the number of subjects needed for a thinking aloud test

JAKOB NIELSEN

SunSoft, UMTV 19-107, 2550 Garcia Avenue, Mountain View, CA 94043-1100, USA

(Received and accepted 28 March 1994)

Two studies of using the thinking aloud method for user interface testing showed that experimenters who were not usability specialists could use the method. However, they found only 28–30% of known usability problems when running a single test subject. Running more test subjects increased the number of problems found, but with progressively diminishing returns; after five test subjects 77–85% of the problems had been found.

Certainly We Were Not Many
And Yet We Were Sufficient
From the Norwegian National Anthem, B. Bjørnson, 1859

1. Introduction

The thinking aloud[†] method has long been employed by usability engineering experts as one of the principal methods for improving user interfaces through user testing (Lewis, 1982; Jørgensen, 1989; Nielsen, 1993). Even so, the available evidence indicates that many current development projects do not take advantage of the method. For example, Milsted, Varnild and Jørgensen (1989) found that only 21% of Danish software developers knew about thinking aloud and that only 6% actually used it. No other systematic evaluation methods were used by the developers, whereas more informal methods such as reviews and heuristic evaluation (Nielsen & Molich, 1990; Nielsen, 1994) were used widely. The main reason that more than two thirds of the developers who know the thinking aloud method still do not use it seems to be that they believe that it is too complex and expensive to use, requires special video taping laboratories, etc.

The "discount usability engineering" philosophy (Nielsen, 1989, 1990, 1993) aims at increasing the use of usability methods by reducing their perceived cost and complexity. The complexity of thinking aloud can be reduced by noting that special usability laboratories with fancy video taping equipment are not really needed to benefit from the method. Also, experimenters do not need to be highly skilled specialists (Nielsen, 1992*a*) but can be ordinary computer scientists with a small

[†] A simplified thinking aloud test involves having a test user operate an interface to perform a set of pre-defined tasks while being asked to "think out loud". By listening in on the user's thoughts, the experimenter can pinpoint misconceptions and other usability problems as they occur in the interaction. Advanced thinking aloud methodology involves formal protocol analysis (Ericsson & Simon, 1984) of a recording of the user's utterances and actions.

amount of training in the running of thinking aloud tests.† It is even possible to have designers serve as experimenters for tests of their own interfaces (Wright & Monk, 1991), thus saving the time that would otherwise be required to have a neutral experimenter learn the system. The present paper aims at reducing the cost of thinking aloud by investigating the trade-offs between the number of subjects in an experiment and the proportion of usability problems found in the interface. For the purposes of the present analysis, we are only concerned with qualitative studies that aim at disclosing usability problems in a design. For quantitative studies aimed at collecting usability measures, one would still need to use sufficiently many subjects to narrow the confidence intervals to an acceptable level.

As a user test method, thinking aloud can be used in several stages of the usability engineering lifecycle (Nielsen, 1992*b*, 1993), from the early experimental prototype stage to almost finished products that need to be polished for release. For many such tests, one is not necessarily interested in exhaustive testing, since the user interface is scheduled to undergo changes as the project progresses. Also, one is often satisfied with qualitative tests, since it is not always necessary to know *how much* better a design is, as long as one is satisfied that it is better than the previous version.

2. Method

Two sets of experiments were conducted using the same method. In each case, a number of computer science students who were taking a class in user interface design were given three hours of lectures on the thinking aloud method and were then asked to run an experiment on their own. Two different (but similar) groups of students were used in the two experiments. The students were asked to test the user interface of a given application but were otherwise free to design their experiment (including test tasks and instructions to the subjects) as they saw fit. Test subjects were two additional (and different) groups of people (24 and 30, respectively). Each experiment used different test subjects.

The experimenters were asked to write reports on their experiments and to list the usability problems they had observed. These reports were then scored for usability problems mentioned and the complete set of usability problems for each application was defined as the union of the problems from the individual reports. There is some possibility that there might be additional usability problems that were not encountered in any of the experiments, but given the fairly large total number of test subjects in the two studies it is likely that any really serious problem has been found at least once and thus included in the aggregate list.

The following two sections describe the systems tested in the two experiments and list the usability problems found during the tests. The complete lists of usability problems were derived by adding up all the individual lists of usability problems from each thinking aloud subject. This means that the aggregate lists presented in

† Please note that usability laboratories and video taping are needed for advanced uses of thinking aloud, and that the studies of non-specialist experimenters (Nielsen, 1991) do indicate that the use of improved methodology leads to better results. The "discount usability engineering" approach acknowledges these facts but emphasizes that there are still large benefits to be gained even with a suboptimal methodology.

TABLE 1
Usability problems in a popular commercial word processor

Usability problem	Frequency with which problem was found with one subject
1. Arrow keys do not work even though they are on the keyboard	42%
2. `cut/copy/paste` hard to learn	42%
3. Changing to a new document hard	33%
4. Scary alert message: change all cannot be undone	29%
5. Two-cursor problem	29%
6. Menu fixation: not seeing the ruler as a place for making formatting changes	25%
7. Confusing menu-based justification commands and ruler-based justification	25%
8. Selected text disappears after `insert ruler` or `insert new page` command	25%
9. Hard to distinguish small markers for margin and tabs in the ruler	21%

Tables 1 and 2 are based on the total number of subjects (24 and 30, respectively) and are therefore probably reasonably complete. Of course, there is never any way to know whether some additional usability problem would have surfaced if one additional subject had been run, but both common sense and the mathematical model presented later in this paper would indicate that this would be unlikely to happen.

The descriptions of the usability problems in the Appendix are intended to document the meaning of the term "usability problem" in the statistical treatment

TABLE 2
Usability problems in a shareware outliner

Usability problem	Frequency with which problem was found with one subject
1. No menu command for "insert new section"	57%
2. Three different `Edit` menus	47%
3. Confusing terminology for operating on sections	40%
4. Mixture of botanical "tree" metaphor and literary "section" metaphor	37%
5. No menu command for promoting/demoting sections	37%
6. Two-cursor problem	27%
7. *Boing* as error message	23%
8. Selection of subtrees is non-standard	23%
9. `Clear tree` command evokes scary alert message	20%
10. Arrow keys have mixed use for navigation and as function keys	20%
11. `Find` is case sensitive	17%
12. Manual talks about CONTROL-TAB instead of COMMAND-TAB	13%
13. Not clear where `Paste tree` will insert subtree	13%
14. Copyright notice flashed on screen at startup	13%

in later parts of this paper. A deeper understanding of the individual problems would be of vital interest for an interface designer charged with the iterative design of the two interfaces, but readers who are pressed for time may want to study the data in the tables and not read the Appendix.

Simply stated, a usability problem is any aspect of a user interface that is expected to cause users problems with respect to some salient usability measure (e.g. learnability, performance, error rate, subjective satisfaction) and that can be attributed to a single design aspect. Thus, having users say that the interface stinks would not be considered a usability problem, though it would certainly be a problem for the product. Having users say that they dislike a blinking field or observing that users have difficulty understanding a certain menu option would be considered usability problems.

3. Experiment 1: a commercial word processor

The first application was version 4.5 of a famous commercial word processor released by a respected computer company. It was tested with a total of 24 thinking aloud subjects. This interface had presumably already been through an extensive iterative development process and could be expected to contain relatively few glaring usability problems. The word processor ran on a personal computer using a graphical user interface, and relied on the mouse for most navigation and selection.

In fact, nine usability problems were identified, as shown in Table 1 and described further in the first list in the Appendix.

4. Experiment 2: a shareware outliner

The second application to be tested was version 2.2 of a shareware outliner (an application for manipulating and reorganizing nested hierarchical outlines of, say, articles or books). It was tested with a total of 30 thinking aloud subjects. This interface was developed by a single hacker and might be expected to contain a somewhat larger number of usability problems than the word processor tested in experiment 1. The outliner ran on a personal computer using a graphical user interface, and relied on the mouse for many operations.

In fact, fourteen usability problems were identified, as shown in Table 2 and discussed in detail in the second list in the Appendix. Considering that the outliner was a considerably smaller piece of software than the word processor, it seems that it was indeed inflicted with usability problems to a larger degree than the word processor.†

† Note, however, that current usability engineering theory has no way to predict usability difficulties in an application on the basis on underlying measures of the size or complexity of the functionality of the application. Intuitively, it just seems that a more complex system would be likely to have more usability problems than a less complex system, assuming that the same amount of usability talent and effort had been invested in both development projects.

5. Discussion

It is immediately obvious from Tables 1 and 2 that it is not sufficient to run a single subject in a thinking aloud test. Many of the usability problems have very small probabilities of being found, and the average number of problems found is only 30% for the word processor and 28% for the outliner.

Luckily, this problem can be solved by running more than one test subject, since different problems are found in the tests of different subjects. There are two possible reasons for this effect: different subjects might encounter different problems, or experimenters might make different observations during different tests. This study has not investigated these underlying issues but my intuition is that both explanations probably hold to some extent.

In any case, the empirical evidence does show that different problems are found when running different subjects. By combining reports from several thinking aloud studies, one can therefore find more problems than found during any of the individual tests. For the following discussion, I will assume that it does not matter how many times a usability problem has been found as long as that number is greater than zero. This assumption is reasonable during iterative design of a user interface where the main need is to build a "bug catalog" listing usability problems to be addressed in the next version. Given that one can never fix the complete list, there is also a need to prioritize the usability problems. Therefore there is reason to count the number of occurrences of the individual problems to get an idea of their relative frequency, but that issue will not be addressed here.

Figure 1 was generated by selecting random subsets of n test subjects for $n = 1$–15. For each subset cardinality, ten thousand random subsets were generated, and Figure 1 shows the average number of usability problems found by these subsets of test subjects. The number of problems found is obviously dependent on the actual subjects included in a subset. After the fact, it would be easy to select the subset of n subjects that found the most problems. It is probably also possible to choose subjects in advance to maximize the variety in usability problems they encounter and thus minimize the number of subjects needed to find each problem at least once. One

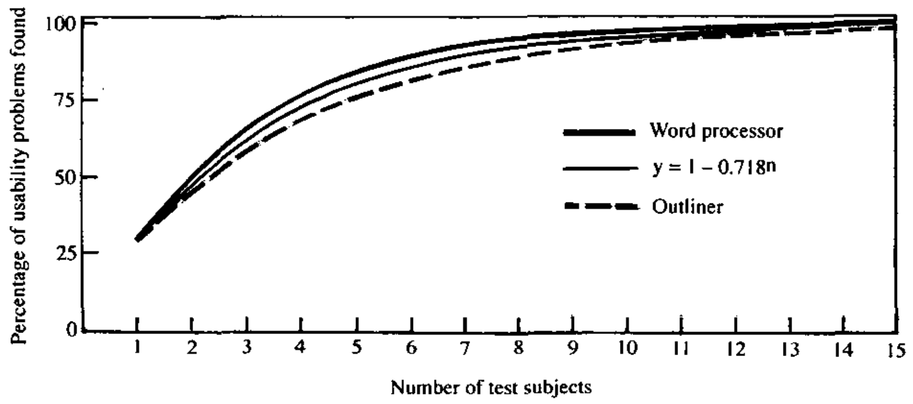

FIGURE 1. The mean percentage of usability problems found by using various numbers of test subjects in a thinking aloud study. The two thick lines show empirically derived numbers and the thin line shows a numerical model.

TABLE 3
Proportion of usability problems found by running various numbers of subjects in a thinking aloud test (combined average for the word processor and the outliner)

No. of subjects	Problems found	Increase due to last subject
1	29%	29%
2	49%	20%
3	63%	14%
4	73%	10%
5	81%	7%
6	86%	5%

likely approach would be to choose subjects with very different backgrounds and previous computer experience. Figure 1 and the following discussion, however, both assume random choice of subjects. The average number of subjects needed to find a usability problem was 3.2 for the word processor and 3.7 for the outliner.

Figure 1 shows the proportion of usability problems found by combining the results from several subjects. There are slight differences between the curves, but they are mostly similar. For both curves, the first subject gives the most information, the next somewhat less, with regularly diminishing marginal gain as more subjects are run. For these cases, half the problems were found with two subjects, three quarters with four. Table 3 shows the proportion of problems found for various numbers of subjects and the benefit from adding the last test subject.

The empirically derived curves in Figure 1 showing the number of problems found for increasing numbers of subjects fit very closely to the Poisson model given in equation (1) of Figure 2. The parameter λ indicates the proportion of the so-far undiscovered problems that will be found during the test of the next subject. The square of $(1-\lambda)$ thus indicates the proportion of undiscovered problems that will still be undiscovered after two additional subjects. A least-squares estimate is $\lambda = 0.282$, and using this value, we find that $(1-\lambda)^2 = 0.515$, indicating that every time one runs two additional subjects, about half of the remaining problems will be found. This result can be used to determine when to stop a sequence of experiments that is intended to discover all usability problems. When only a single usability problem has been found with the last two subjects, it is likely to be because the remaining number of problems is zero or one, and when *no* new problems were found for the last two subjects, then all problems have probably been found.

By modifying equation (1) by substitution from definition (2), we arrive at equation (3) which can be used for early predictions of the total number of usability problems in an interface (see Figure 2). Of course, the parameter λ has to be known and there is no guarantee that it will stay the same for other interfaces. In fact, in a number of projects surveyed, λ varied from 0.12 to 0.48 (Nielsen & Landauer, 1993), so it is essential to derive estimates of λ from local experience. Using the value of

TABLE 4
*Standard deviations of the guesses of the total number of usability problems expressed as percentages of that number. For each of the two systems, the left column shows standard deviations of guesses based on the **sum** of usability problems found. The right column shows standard deviations of guesses based on the number of **different** problems found*

	Word processor		Outliner	
No. of subjects	Sum of problems	Different problems	Sum of problems	Different problems
1	62%	62%	42%	42%
2	43%	40%	29%	26%
3	34%	29%	24%	20%
4	29%	22%	20%	15%
5	25%	17%	17%	13%

$\lambda = 0.282$, Table 4 shows the standard deviation of the guesses of the total number of usability problems made after knowing the results from varying numbers of subjects. The table shows two ways of using equation (3). First, one can count the total number of usability problems observed (counting any given problem several times if it is observed for several subjects) and take the average per subject to arrive at a more reliable estimate of the number of problems for a single subject, inserting that value in equation (3) with $n = 1$. Second, one can count the total number of different problems observed and insert that value in equation (3), using the actual value of n. Table 4 shows that the latter approach gives better guesses (smaller standard deviations).

6. Speculation

Both experiments tested interfaces that had already been released in at least a second version. Assuming that products do get better with each release, it is likely that interfaces at an earlier stage of the development lifecycle would have more usability problems and that these usability problems might be more glaring than the ones found in these experiments. These considerations might lead to the speculation that user testing of early interface designs would find more usability problems per

$$\text{Proportion of problems found with } n \text{ subjects} = 1 - (1 - \lambda)^n \qquad (1)$$

$$\text{Problems found with } n \text{ subjects} = \text{Total problems} \times \text{Proportion found with } n \text{ subjects} \qquad (2)$$

$$\text{Total problems} = \frac{\text{Problems found with } n \text{ subjects}}{1 - (1 - \lambda)^n} \qquad (3)$$

FIGURE 2. Empirical model for predicting the total number of usability problems in an interface.

test subject and that the trade-off curve for such tests would therefore be even more in favor of using a small number of subjects. Assuming that an iterative design approach is taken, there is even more reason to recommend the use of a fairly small number of subjects for user testing of each iteration. Since a new iteration often introduces new usability problems and sometimes does not even fix the old problems properly, it is likely that the final usability will be enhanced more by having a large number of iterations (each of which are only tested to find a reasonably large proportion of the usability problems) than by testing a small number of iterations using large numbers of subjects for each iteration.

For real-life, industrial user testing, it is obviously more important to find the major usability problems than to find the minor ones since there will never be time enough to correct all the problems anyway. Unfortunately, the definition of "major usability problem" is not completely clear, but it would certainly include parameters such as (a) how many users encounter the problem, and (b) how hard it is for the individual user to overcome the problem once it is encountered.

If these two parameters are accepted as indicators of the severity of a usability problem, then it follows by definition that a smaller number of test subjects will be needed to find the major usability problems than are needed to find all the usability problems, including the minor ones. From parameter (a) it follows that the major problems are statistically more likely to occur than the minor problems, and from parameter (b) it follows that the experimenter will be less likely to overlook a major problem than a minor problem. Thus, both aspects of the definition increase the probability that an experimenter will identify a major usability problem even with a small number of subjects.

Empirical support for this conjecture comes from Virzi (1990). He had the experimenters in a thinking aloud study rate the usability problems on a scale from one (minimal effect on usability) to seven (severe detriment to usability). He found that the severe problems could be found with significantly fewer subjects than the less severe problems. For example, the average number of subjects needed to find a major problem (rated as having severity six or seven) was 1.7 compared to the 3.4 subjects needed on the average to find a medium problem (severity three to five) and the 5.1 subjects needed for minor problems (severity one or two).

A final reason to recommend the use of a small number of test subjects is a belief that the experimenters will be able to observe more from each subject as they get more experienced in planning and running thinking aloud studies. The experimenters used for the two tests reported in this paper were very inexperienced and were in fact running their very first user test of any kind. It is likely that they would be able to improve their methodology as they got more experience. Indeed, a previous study (Nielsen, 1992*a*) showed a positive correlation between the degree to which experimenters followed the recommended methodology and the number of usability problems they found in a thinking aloud study of an interface.

7. Conclusions

The data from this study, as shown in Figure 1, show that thinking aloud studies do not need a large number of subjects to be successful at identifying usability problems in a user interface. In the two reported here, it was possible to find about 75% of the usability problems by running only four to five subjects. Assuming that the goal is to identify usability problems for correction in future iterations, it does not seem to be worth the trouble to run more than six or seven subjects, since the value of additional subjects falls off exponentially.

Intuitively, it seems to be a reasonable goal to aim at finding about 75% of the usability problems in an interface because:

- it will be very expensive to find all the problems
- the next design iteration will probably introduce some new problems anyway
- the true usability catastrophes are likely to be among the first problems found.

Furthermore, more experienced experimenters will probably be able to do better than the students who served as experimenters in the two studies reported here. So my final recommendation would be to **plan for 4 ± 1 subjects in a thinking aloud study,** with the final number of subjects determined by:

- the skills and experience of the experimenter
- the number of iterations planned for the design
- the financial or other impact of the use of the system.

In early 1994, I surveyed 36 usability specialists from a variety of companies who had conducted an average of 9.3 usability tests each. The average number of test subjects they used for a test was 8.8, and only 35% of the respondents were using between three and six test users per test. Thus, it would seem that there is great potential for cost savings if development projects would run smaller user tests (and then do more of them). Even so, there are obviously still some projects for which "deluxe" usability testing is appropriate, including projects where a single remaining usability problem would be catastrophic or life-threatening.

The author would like to thank Tom Landauer and the anonymous *IJCHS* referees for valuable comments on previous versions of the manuscript.

References

Brooks, F. R. (1988). Grasping reality through illusion: Interactive graphics serving science. *Proceedings of the ACM CHI'88 Conference*, pp. 1–11, Washington, DC, 15–19 May.

Ericsson, K. A. & Simon, H. A. (1984). *Protocol Analysis: Verbal Reports as Data.* Cambridge, MA: MIT Press.

Jørgensen, A. H. (1989). Using the thinking-aloud method in system development. In G. Salvendy & M. J. Smith, Eds. *Designing and Using Human–Computer Interfaces and Knowledge Based Systems*, pp. 743–750. Amsterdam: Elsevier.

Lewis, C. (1982). *Using the 'thinking-aloud' method in cognitive interface design.* Research Report **RC-9265,** IBM T. J. Watson Research Center, Yorktown Heights, NY, USA.

Milsted, U., Varnild, A. & Jørgensen, A. H. (1989). Hvordan sikres kvaliteten af brugergrænsefladen aie systemudviklingen (Assuring the quality of user interfaces in system development, in Danish). *Proceedings of NordDATA'89 Joint Scandinavian Computer Conference*, pp. 479–484, Copenhagen, Denmark, 19–22 June.

NIELSEN, J. (1989). Usability engineering at a discount. In G. SALVENDY & M. J. SMITH, Eds. *Designing and Using Human–Computer Interfaces and Knowedge Based Systems*, pp. 394–401. Amsterdam: Elsevier.

NIELSEN, J. (1990). Big paybacks from 'discount' usability engineering. *IEEE Software*, **7**(3), 107–108.

NIELSEN, J. (1992*a*). Evaluating the thinking aloud technique for use by computer scientists. In H. R. HARTSON & D. HIX, Eds. *Advances in Human-Computer Interaction*, Vol. 3, pp. 69–82. Norwood, NJ: Ablex.

NIELSEN, J. (1992*b*). The usability engineering life cycle. *IEEE Computer*, **25**(3), 12–22.

NIELSEN, J. (1993). *Usability Engineering*. Boston, MA: Academic Press.

NIELSEN, J. (1994). Heuristic evaluation. In J. NIELSEN & R. L. MACK, Eds. *Usability Inspection Methods*, pp. 25–62. New York: John Wiley.

NIELSEN, J. & LANDAUER, T. K. (1993). A mathematical model of the finding of usability problems. *Proceedings of the ACM INTERCHI'93 Conference*, pp. 206–213, Amsterdam, The Netherlands, 24–29 April.

NIELSEN, J. & MOLICH, R. (1990). Heuristic evaluation of user interfaces. *Proceedings of the ACM CHI'90 Conference*, pp. 249–256, Seattle, WA, 1–5 April.

VIRZI, R. A. (1990). Streamlining the design process: Running fewer subjects. *Proceedings of the Human Factors Society 34th Annual Meeting*, pp. 291–294, Orlando, FL, 8–12 October.

WRIGHT, P. C. & MONK, A. F. (1991). A cost-effective evaluation method for use by designers. *International Journal of Man-Machine Studies*, **35**, 891–912.

Paper accepted for publication by the Editor, Professor B. R. Gaines.

Appendix: descriptions of the usability problems

This appendix provides somewhat more detailed descriptions of the usability problems than can be accommodated within the main paper. These descriptions are not of great importance for readers who are only interested in the main point with respect to how many subjects to use in a qualitative usability test. The descriptions are intended for readers who are interested in the methodological issues related to the question of what constitutes a usability problem and for readers who want to assess the relative severity of the various usability problems for themselves.

The first list describes the nine usability problems found in the commercial word processor.

Problem 1. The arrow keys had no effect in this word processor even though they were present on the keyboard. This problem was corrected in version 5 of the application.

Problem 2. Users had problems learning the standard cut/copy/paste commands. First, the commands only worked if something had been selected. Second, the copy command gave no feedback, leading some users to wonder whether it had any effect.

Problem 3. Because this version of the program only supported one open document at a time, it was not possible to open a new document before the previous document had been closed. Closing a document is a very indirect action if one has the goal of opening a document, so many users tried the open command first. Upon seeing that it was grayed out in the menu, they often realized that it was disabled for some reason but they were not always able to discover this reason. Assuming that it was indeed necessary to restrict the system to a single open document, it might have been better to allow users to select the open command and then present them with a dialog box stating that the current document would have to be closed first and asking them whether they want this done or whether they want to cancel.

Problem 4. Due to yet another underlying functionality limitation, the word processor could not support undo for the global replacement operation. The interface correctly informed users about the limitation but did so using such scary language that several users did not dare use this feature at all.

Problem 5. The "two-cursor problem" (Brooks, 1988) of having one cursor indicating the text insertion point and another indicating where the mouse is pointing. Some users assumed that they would be typing at the mouse pointer and were confused when the text appeared elsewhere on the screen. One subject tried to move the insertion point to a new location by direct manipulation (clicking on it with the mouse pointer and trying to drag it to a new location).

Problem 6. Some users suffered from menu fixation and believed that the selection of menu commands was the only way to operate the interface. In fact, some formatting operations such as margin indentation could only be activated by direct manipulation of markers on a graphical "ruler".

Problem 7. The word processor had two different sets of justification commands. One set was located in the menus and only changed the justification of the currently selected lines. Another set was located on the graphical ruler and changed the justification of all lines until the next ruler, thus having a more global effect (especially since most novice users only have a single ruler in a document). The difference between these two sets of commands was not clear to some users. Sometimes they would use a menu-based justification command and be quite surprised when the operation had no effect on subsequently typed text.

Problem 8. The commands "insert ruler" and "insert new page" worked by replacing the selected text with the special object indicated by the command (a new ruler or a top-of-page marker). This may have been a local extension of the cut-and-paste paradigm, but several users did not expect this to happen and got very scared when some of their text disappeared as a result of these commands. It might have been better to simply insert the new special object in front of the selected text rather than replacing it.

Problem 9. The graphical ruler had some small markers used to set margins and tabulators, with different markers used for slight variations in functionality such as setting the left margin of the first line of a paragraph versus setting the left margin of the rest of the paragraph. Some users had difficulties in distinguishing between these markers.

This second list describes the fourteen usability problems found in the shareware outliner.

Problem 1. Users searched in vain for a menu command to insert a new section (element) in the outline. In fact, new elements were inserted "simply" by pressing the return key but many users had a hard time discovering this. One reason for this problem is that the return key was seen as a text processing key (to insert white space in a document) and not as a function key to create new objects.

Problem 2. The outliner had three different edit menus: one for cut-and-paste of actual text, one for cut-and-paste of entire sub-tree structures of nested sections, and one for cut-and-paste of the contents of a kind of database of text strings called a textstore. Users were often confused by this deviation from the standard of having a single, generic edit menu that operates on all kinds of objects.

Problem 3. The terminology used in the menu commands for operating on sections was somewhat confusing. In addition to the headings of the various sections in the outline, the system allowed the user to operate on an additional text string for each section. These additional strings were normally not shown but could be displayed by the command "open section". Users sometimes confused this command with the command to expand the hierarchy to make the subsections under the current section visible. This command was called "expand subsections" and its inverse command was "hide subsubsections" (note, *two* "sub").

Problem 4. The program used mixed metaphors for the hierarchical outline and this sometimes caused confusion. One metaphor was a literary one of sections and subsections (used in some commands) and another was a botanical one of trees and subtrees (used in other commands to operate on a section and its subsections as a whole). Also, some users misinterpreted the term "tree" as referring to the entire outline rather than to the currently selected subtree. One user interpreted the menu command "cut tree" in analogy to cutting timber and was afraid to use it because it seemed too drastic.

Problem 5. Several users searched for a menu command to promote subsections to sections or to demote sections to subsections. There were no explicit commands to accomplish this goal and users were expected to use the left and right arrow keys as function keys. Note that this problem and problem 1 seem to be similar to the menu fixation problem reported as problem 6 of the word processor.

Problem 6. The two-cursor problem (discussed as problem 5 of the word processor) was also observed here.

Problem 7. The sound "boing" was used as practically the only error message throughout the system. This sometimes annoyed users and sometimes caused further problems because they did not understand the underlying error situation. Also, disabled menu commands were not grayed out but caused further boings when selected.

Problem 8. Selecting entire subtrees of the hierarchy was done by placing the insertion cursor in the heading of the top-most element of the subtree. There was no visual indication of the extent of the selection and the actual operation was the same as that used for selecting text. Actually, a subtree would be selected even if no text was selected when the insertion point was positioned between two text characters.

Problem 9. In a problem similar to that reported as problem 4 of the word processor, the "clear tree" command gave a scary alert message stating that it could not be undone. Users were often so frightened by this message that they abandoned any attempt to use the command even when it would have been correct of them to do so.

Problem 10. The arrow keys had two mixed uses: the up/down keys were used for navigation (moving the insertion point one line up or down) whereas the left/right keys were used as command keys (to promote or demote a section in the hierarchy). Users sometimes confused these two uses of a set of keys that are normally very strongly grouped.

Problem 11. The "find" command was strictly case sensitive and would not find, say, a word starting with an upper case character if the search string was entered using all lower case characters. This sometimes led users to assume either that the word did not exist in the document or that the find command did not work.

Problem 12. The manual used the term "CONTROL-TAB" to refer to holding down the command key while pressing the tabulator key. Some keyboards actually had a key labeled "control" which was *not* the command key and some users had severe problems finding the correct key to hold down.

Problem 13. Some users were confused where the "paste tree" command would insert the subtree. Normally, a paste command inserts the content of the clipboard at the insertion point, but the "paste tree" command inserted a subtree *below* the section heading containing the insertion point.

Problem 14. When starting the program, a window with a copyright notice would flash on the screen for a brief moment. Some users were frustrated at not being able to read the full contents of the window and feared having missed some important information.