SESSION VI UNDERSTANDING PROGRAMMING

Cyndi McDaniel, *Chair*

The influence of structure on computer program comprehension

SANDRA L. NEWSOME

Rensselaer Polytechnic Institute, Troy, New York

and

IRVIN R. KATZ

Carnegie-Mellon University, Pittsburgh, Pennsylvania

Previous empirical studies that have relied on debugging (Lucas & Kaplan, 1974) and memory (Sheppard, Borst, Curtis, & Love, 1978) as measures of computer program comprehension have led to inconsistent results and conclusions regarding the benefits of structure. The present experiment utilizes a new technique for measuring comprehension that is sensitive to contextual factors that help the programmer to anticipate upcoming lines of a program. Subjects in this experiment were told the general purpose of a program written in English and asked to anticipate the subsequent lines. After the subjects entered their anticipated line, the actual line of the previously written program was shown. Subjects used the lines of the program shown as a contextual guide for their next anticipation. The percentage of correctly anticipated lines was greater for structured programs than for unstructured programs. However, the advantage of structure was more pronounced for long, complex programs than for short, relatively straightforward programs.

Although textbooks on programming and journal articles by computer scientists (e.g., Schwartz, 1976) frequently indicate that a structured style of programming (i.e., single entry and single exit from a loop) is an absolute necessity for comprehensible programs, the empirical literature within the domain of a human-computer interaction regarding this issue is far from conclusive (Lucas & Kaplan, 1974; Sheppard, Borst, Curtis, & Love, 1978). Inconsistent results in the empirical literature can be explained in part by the difficulty of measuring the cognitive variable of comprehensibility and the difficulty of separating the effects or constraints of previously learned computer languages on comprehension strategies. The purpose of the present study was to examine a new technique for measuring comprehensibility in an environment that reduced the influence of the prior learning of a particular computer language.

The difficulties of finding a sensitive measure of the comprehension of computer programs, as well as the inconsistencies in results in the literature, are most easily

examined through an analysis of previous attempts to measure the influence of structure. For example, Lucas and Kaplan (1974) investigated the influence of structure by asking students in a PL/I course to compose and debug programs with or without the use of GOTO loops. Students required to compose the program without GOTO loops took considerably longer than students who were allowed to use the loops. However, debugging of the programs was more rapid for versions of the program without GOTO. Since comprehension is assumed to be necessary for efficient debugging, one could conclude from this study that structure, in the form of versions of the program without GOTO, aided comprehension.

However, other investigators have reached the opposite conclusion. For example, Sheppard et al. (1978), using a study-recall method, found that structure had little influence. In their study, experienced Fortran programmers studied a program for a brief amount of time and then attempted to recall as much of the program as possible after it had been removed. Comprehension of the program in this study was measured by the number of lines correctly recalled. Recall was measured for three types of Fortran programs, which varied both in the number of exits from loops and in the degree to which the loops stylistically deviated from what might be expected in For-

We would like to thank Al Bottnar and John Schlosser for helping prepare the stimulus materials and testing the subjects. We also are grateful for Jackie Schrier's assistance in scoring and testing of subjects. Please address reprint requests to Sandra Newsome, Psychology Department, Carnegie 305, Rensselaer Polytechnic Institute, Troy, NY 12180-3590.

tran. The results of the study indicated that recall of the programs that had only a single forward loop (structured) was the same as that of programs that permitted two exits from a loop but were stylistically natural for Fortran (unstructured).

We would argue that the contrast in the results of the Lucas and Kaplan (1974) study and the Sheppard et al. (1978) study was due in part to the relatively insensitive measures of comprehension used and to the influence of the type of computer languages tested. Consider first the two methods of measuring comprehension: debugging and memory. As stated before, comprehension does indeed seem to be a necessary precursor for debugging. However, it is unclear whether an entire program needs to be understood in order to find all types of bugs or whether all debugging strategies lead to the same level of comprehension. For example, if one uses a hand-simulation method of debugging, only knowledge of the computer language and a general understanding of the program is required. However, if debugging proceeds by an analysis of the output of the program, a relatively high level of comprehension is required. It is also the case that some bugs are easier to detect than others (Gould, 1975). Consequently, different depths or levels of comprehension (Craik & Lockhart, 1972) may be achieved in the debugging process. For these reasons, among others, we argue that debugging is an extremely difficult to interpret measure of comprehension.

Memory also suffers from shortcomings regarding sensitivity as a measure of comprehension. Not only has Adelson (1981, 1984) shown that the amount of a computer program remembered is strongly influenced by the level of computer expertise of the subjects, but her results suggest that what is remembered is considerably different for novices and experts. Since in Sheppard et al.'s (1978) study, the measure of comprehension was the number of lines recalled and no analysis of what was recalled was provided, one cannot help but wonder what level of comprehension was measured. If subjects recalled mostly assignment statements, should we assume that subjects comprehended the semantics of the program?

A second, more general, problem with memory is that it is not necessarily isomorphic with comprehension. Variables that influence memory do not necessarily influence comprehension in a similar manner. For example, the von Restorff effect (Crowder, 1976) refers to the fact that a highly salient or unusual word appearing in the middle of a serial list will be more likely to be recalled than other words in the middle of the list. A similar phenomenon in computer programs is easily imagined. If an algorithm is encountered that is particularly novel or even particularly difficult to understand (and thus requires more study), it is probably more likely to be recalled. However, we would certainly not want to conclude that recall in this case necessarily represents a passage that is easy to understand.

In addition to the problems associated with the measures of comprehension in these studies, the results of both

of the studies may have been influenced by the particular computer languages the investigators chose to study. When a particular computer language is used in studies of comprehension, it requires a subject population that is trained to some extent in that language. If the computer language involved tends to be written or taught in a structured or unstructured manner, then it is possible that the subjects have already learned a particular style of programming that will influence all measures of comprehensibility. It could be argued that the programmers in Sheppard et al.'s (1978) study had learned an unstructured programming style when they learned Fortran, whereas programmers in Lucas and Kaplan's (1974) study had acquired some aspects of structured programming in learning PL/I. This difference in previously learned programming style would certainly account for the differences in results for the two studies, but it would also indicate that the implications of the studies are limited only to the languages that were studied. In other words, the general issue of how structure influences comprehension in computer programs is unresolved.

The present experiment avoided the difficulties of interpretation encountered in previous attempts to measure the influence of structure on comprehensibility by utilizing a more immediate measure of comprehension for programs that were translated from computer code to natural English. The measure of comprehension used in our experiment required subjects to attempt to anticipate lines of English computer programs using previous context as a guide. This technique of measuring comprehension is based on the assumption that "good" context helps the comprehender to anticipate upcoming text and subsequently facilitates comprehension.

Although this technique has not been applied to computer program comprehension, many cognitive psychologists have relied on this assumption in attempting to determine the effects of meaning or context in natural language (e.g., Neely, 1976; Schvaneveldt & Meyer, 1973). Consider the lexical decision task. In the lexical decision task, subjects are visually presented with a letter string and required to determine whether the letter string is a legitimate English word. Numerous studies (e.g., Neely, 1976; Schvaneveldt & Meyer, 1973) have found that subjects are faster to respond to a word if it is preceded by a word that is related in meaning than to a word preceded by an unrelated word. For example, subjects are faster to say *nurse* is a word if it is preceded by *doctor* than if it is preceded by an unrelated word, such as *butter*. One way of interpreting these findings is to suggest that the context word *doctor* helps the subjects to anticipate the target word *nurse*.

This technique of measuring the usefulness of context was applied to computer program comprehension in order to determine whether structured context in a program helped subjects to anticipate correctly the upcoming lines. In our experiment, subjects were told the overall purpose of a computer program and were asked to anticipate the first line of the program. After the subjects entered their

responses, the first line of the previously written computer program was shown to them. From the line shown to them, subjects were then asked to try to anticipate the second line and so on for the entire program. We predicted that if structured context facilitated the comprehension process, subjects having structured context as a guide would be more accurate in their anticipations than would subjects having only unstructured context as a guide.

As mentioned earlier, the present experiment used English language programs that were translated from the computer code. The advantage of using computer programs that were not written in a specific computer language was that subjects with a variety of computer backgrounds were able to participate in the experiment. In addition, structured context dictated by the syntax of a particular computer language was avoided. Consequently, the chances that prior learning and/or knowledge of a particular computer language biased the results were considerably reduced.

METHOD

Subjects

Forty undergraduates from Rensselaer Polytechnic Institute volunteered to participate in the experiment. Some of the subjects received extra credit in a psychology class in exchange for their participation. Overall, the subjects could be judged as being relative computer experts. The mean number of different computer languages known by the subjects was 4. The number of different computer languages known by students ranged from 0 to 10.

Test Materials

All subjects were asked to anticipate the lines of two different computer programs that were written in English. To ensure that the English-language programs represented legal executable programs, all versions of the programs were initially written in code, tested, and then translated into the English equivalent.

One of the programs (sort program) alphabetically sorted a list of 20 names and printed the sorted list. The other program (game program) played what is commonly called a high-low game with a user. The program generated a random number between one and 100, and the user was allowed six chances to try to guess the number. After each of the user's guesses, feedback was provided as to whether the guess was high, low, or correct. The sort program was 18 lines long and could be considered more complex than the game program since more loops were necessary. The game program was 10 lines long.

Two versions of each of these programs were written. In one version, the structured version, there was only a single entry and single exit from each loop. In the other version of each program, the unstructured version, there were two methods of exiting from the primary loop.

Design

Since we initially thought the influence of structure would be the same in both programs, we had all subjects

work with both a structured and unstructured program. Subjects that worked with the structured sort program worked with an unstructured game program, and vice versa. In addition, to prevent ordering effects, half of the subjects worked with the game program first and the other half worked with the sort program first.

Procedure

Subjects were individually tested in a session that lasted approximately 1 h. The subject sat facing two computer monitors and two keyboards. Instructions for typing the anticipated line of the program and the general purpose of the program were presented on the monitor at the right of the subject. After the instructions and general description were presented on this monitor, the screen remained blank until the subject began typing his/her input. The subject typed in the responses on the right-hand keyboard. After the subject finished typing the anticipated line, it was erased from the right-hand monitor; the subject then pushed the space bar on the left-hand keyboard, and an actual line of the English computer program appeared on the left-hand monitor. The actual lines of the English computer program revealed in this manner remained visible to the subject throughout the entire time he/she worked on that program. The subject was told to base the next anticipation not on the line he/she had typed, but on the line presented on the left-hand monitor. The alternating process (i.e., the subject's typing the anticipated line and then seeing the actual line) continued until the end of the program, when the general purpose of the next program was described.

Each subject's anticipated lines were stored so that they could be compared to the actual lines of the program at a later time. After the anticipation tasks, subjects completed a questionnaire on their computer background.

RESULTS

For an anticipated line to be counted as correct, two scorers (one programming naive) had to agree that the entire anticipated line represented a correct paraphrase of the actual line of the program. Changes in variable names and changes in the order in which arguments were mentioned were only scored as incorrect if they differed from the logic of the actual programs. The percentage of lines correctly anticipated for the structured and unstructured programs was calculated in order to compare performance on the two types of programs.

Because the data did not appear to be normally distributed, nonparametric statistical tests were used to evaluate the hypothesis. A Wilcoxon signed rank test for matched units (Bradley, 1976) indicated that the percentage of correctly anticipated lines was not significantly greater for structured programs than for unstructured programs (28.8% vs. 23.2%, respectively; $W+ = 469$, $p > .05$). An examination of the raw data indicated that this 5.6% difference was not significant because of the surprisingly large differences in performance on the two types of programs. The sort program (i.e., the longer of

Table 1
Mean Percentage of Correctly Anticipated Lines

Structured Version	Unstructured Version	Difference
	Sort Program	
23.6	16.4	7.2
	Game Program	
33.5	30	3.5

the two programs) was more difficult to anticipate than the game program for both the structured and unstructured versions, as can be seen in Table 1.

Because it seemed likely that the benefits of structured context were obscured by general differences in the two types of programs, two other analyses were performed that examined each type of program separately in a between-subjects comparison using the Wilcoxon rank sum test (Bradley, 1976). For both the sort and game programs, the mean percentage of correctly anticipated lines for the structured versions was significantly greater than for the unstructured versions ($W_n = 348.5$, $p < .001$, and $W_n = 416$, $p < .001$, respectively).

DISCUSSION

Overall, the results of this experiment support the conclusion that structure is indeed helpful context and is likely to facilitate the comprehension process, although the magnitude of the benefits may not be the same for all programs. The fact that structure facilitated comprehension more for the sort program than for the game program was not too surprising when compared to similar results in the cognitive literature. As stated earlier, the sort program was longer than the game program and was more complex in terms of the number of loops required. It was also the case that there were a greater number of different ways to program an alphabetic sort than to program the high-low game. Structured context facilitated anticipation more when the task was relatively difficult, as in the sort program, than when the task was relatively simple, as in the straightforward game program. A similar finding in natural language occurs when stimuli are visually degraded in the lexical decision task (Meyer, Schvaneveldt, & Ruddy, 1975). Recognition time for related words is less when stimuli are visually degraded than when they are visually intact. In both cases (i.e., natural language and computer language), it appears that context is most influential when comprehension is difficult.

The results of this experiment offer strong support for what computer scientists have long believed intuitively. Structure, particularly in longer, complex programs, provides contextual constraints that allow the programmer

to anticipate subsequent lines relative to an unstructured context. It is possible that the benefits of structured context are due primarily to the reduction of uncertainty. In other words, there are probably a greater number of different ways to write an unstructured computer program than a structured program. However, a direct evaluation of this explanation has not been performed.

The results of the present experiment also help to resolve the previous inconsistencies in the literature on the issue of structure. The conflicting results found by Lucas and Kaplan (1974) and Sheppard et al. (1978) were likely caused by differences in the particular computer languages studied and by the particular measures of comprehension utilized in those studies. In contrast to techniques used in past studies, the technique used in the present experiment provided a rather direct measure of studying the usefulness of context in programming and allowed a generalization beyond a specific programming language. The measurement technique promises to be a useful tool in future studies for evaluating other contextual factors that are believed to be important in computer program comprehension.

REFERENCES

- ADELSON, B. (1981). Problem solving and the development of abstract categories in programming languages. *Memory & Cognition*, 9, 422-433.
- ADELSON, B. (1984). When novices surpass experts: The difficulty of a task may increase with expertise. *Journal of Experimental Psychology: Learning, Memory & Cognition*, 10, 483-495.
- BRADLEY, J. V. (1976). *Probability; Decision; Statistics*. Englewood Cliffs, NJ: Prentice-Hall.
- CRAIK, F. I. M., & LOCKHART, R. S. (1972). Levels of processing: A framework for memory research. *Journal of Verbal Learning & Verbal Behavior*, 11, 671-684.
- CROWDER, R. G. (1976). *Principles of learning and memory*. Hillsdale, NJ: Erlbaum.
- GOULD, J. D. (1975). Some psychological evidence on how people debug computer programs. *International Journal of Man-Machine Studies*, 7, 151-182.
- LUCAS, H. C., & KAPLAN, R. B. (1974). A structured programming experiment. *Computer Journal*, 19, 136-138.
- MEYER, D. E., SCHVANEVELDT, R. W., & RUDDY, M. G. (1975). Loci of contextual effects on visual word recognition. In P. Rabbitt & S. Dornic (Eds.), *Attention and performance V* (pp. 98-118). New York: Academic Press.
- NEELY, J. H. (1976). Semantic priming and retrieval from lexical memory: Evidence for facilitatory and inhibitory processes. *Memory & Cognition*, 4, 648-654.
- SCHVANEVELDT, R. W., & MEYER, D. E. (1973). Retrieval and comparison processes in semantic memory. In S. Kornblum (Ed.), *Attention and performance IV* (pp. 395-409). New York: Academic Press.
- SCHWARTZ, J. T. (1976). What programmers should know. *Computer Languages*, 2, 21-25.
- SHEPPARD, S. B., BORST, M. A., CURTIS, B., & LOVE, T. (1978). *Predicting programmers' ability to understand and modify software* (Report No. TR-78-388100-3). Arlington, VA: General Electric.