Debugging by Skilled and Novice Programmers

Leo Gugerty and Gary M. Olson

Department of Psychology
The University of Michigan
330 Packard Road
Ann Arbor, Michigan 48104

Abstract

Two experiments investigated expert-novice differences in debugging computer programs. Debugging was done on programs provided to the subject, and were run on a microcomputer. The programs were in LOGO in Exp. 1 and Pascal in Exp. 2. Experts debugged more quickly and accurately, largely because they generated high quality hypotheses on the basis of less study of the code than novices. Further, novices frequently added bugs to the program during the course of trying to find the original one. At least for these simple programs, experts superior debugging performance seemed to be due primarily to their superior ability to comprehend the program.

Introduction

We report two studies of expert and novice programmers working at debugging a program written by someone else. While programmers often work at debugging a program they have written themselves, frequently individuals who are part of programming teams must work on the code generated by others.

Debugging is a central part of computer programming. Programs seldom work when first coded. Many superficial syntactic errors are easy to find, especially with the aid of a compiler or interpreter. But a substantial number of bugs require concentrated problem solving. Though various debugging strategies and tricks are often taught in programming classes, much of the skill of debugging is learned through the experience of writing programs and getting them to run.

What kinds of skills are required to debug a program written by someone else? Obviously, one main component is knowledge of programming constructs. In order to debug a program written by someone else, the first phase is both to understand what it actually does and what it is supposed to do. These two sources of information combine to generate hypotheses as to what the bug is. These hypotheses are checked by editing the code, running the program, and studying the consequences, or by engaging in desk checking.

 ACM 0-89791-180-6/86/0400 - 0171 $00.75

Jeffries (1981, 1982) reported findings from an informal description of debugging behavior. She compared 6 experts and 4 novices on the debugging of short Pascal programs. Her experts were graduate students in computer science, while the novices had just finished their first programming course. Each subject debugged two short Pascal programs, each of which contained a number of bugs. Subjects worked with a print-out of the program and a print-out of output from test runs--they did not use a computer. Experts found more of the bugs and found them faster. Experts spent a large proportion of their time on comprehension of the program, and constructed a much more complete representation of the program. They also remembered details of the program better than novices did. Novices and experts seemed to use similar strategies while debugging, but the experts had much superior comprehension skills. Because of the small number of subjects she tested, no statistical evaluation of her findings was reported.

In our experiments we also compared expert and novice programmers. We defined expertise the same way Jeffries did. In the first experiment, we taught subjects the LOGO graphics language and had them debug several simple LOGO programs. In the second study, expert and novice Pascal programmers debugged a Pascal program.

Experiment 1

Method. The subjects in this experiment were 18 novices and 6 experts. The novices were just finishing their first or second course in Pascal, while the experts were advanced graduate students in computer science.

None of the subjects knew LOGO prior to the experiment. Thus, the first phase of the study consisted of teaching each subject the LOGO programming language and the LOGO graphics programming environment on an Apple II computer. Each subject was taught individually in a session that lasted approximately two hours. Subjects were taught the basic commands of the LOGO language and the basics of the programming environment, and then were asked to write and debut a short LOGO program. The subset of LOGO used in this experiment was very similar to Pascal, which all the subjects knew.

Therefore, the main thing that subjects learned in the training was the set of LOGO graphics commands.

Following training, each subject debugged three programs. The subject's task was to correct the program so it produced the correct graphical output. All debugging was done on-line. Each defective program was read in from a floppy disk, and the subject was given a print-out of the defective program and a drawing of what the program was supposed to draw. A 30-minute time limit was imposed for the debugging of each program. Subjects were not told how many bugs the program contained. Subjects were asked to think out loud while engaged in debugging. The session was also video taped, primarily to record the information that appeared on the computer screen.

There were three different programs, ranging in length from 16 lines to 49 lines. Each program was supposed to draw a simple figure or design on the screen. One bug was put into each program. Three different types of bugs were used: incorrect graphics procedure parameter, missing graphics interface statement, and global/local variable error. Each of the bug types was combined with each of the programs to produce 9 different buggy programs. All of the buggy programs were syntactically correct and produced graphical output when run. An individual subject saw each program and each bug type once, and across subjects all combinations of program and bug type were used equally often. Order of presentation was also counterbalanced.

This first experiment was a preliminary study, and the experts and novices were run at different times. One of the three programs was slightly modified for the experts, and some small procedural changes were made. However, these slight differences do not account for the contrast in behavior between experts and novices in this study.

Results. Since the programs were fairly simple, most subjects were able to find the bugs most of the time. Experts found the bug 89% of the time, while novices found it 72% ($t(22) = 1.62, .05 < p < .10$). However, experts found the bugs much faster than novices. The mean time to find the bug was 7.0 min. for experts and 18.2 for novices ($t(22) = 3.27, p < .01$).

All subjects began each debugging session by studying the program. This included studying both the printed copy we gave them and a listing of the program on the screen, studying the intended drawing, and -- often but not always -- running the program to see what the flawed output looked like. At some point a subject got an idea as to what the bug might be, and tested this idea by editing the program and running it. We used this set of acts -- editing and running -- to operationalize hypothesis testing[1]. Overall, novices tested 5.1 hypotheses per program, while experts tested an average of 1.9. Looking only at those subjects who were successful, novices went through an average of 3.6 hypotheses, while experts went through 1.5 ($t(22) = 2.19, p < .05$). Experts often tested the correct hypothesis on their first try. 56% of the experts' attempts to debug a program were solved with their first hypothesis, whereas novices found the correct hypothesis first only 21% of the time ($t(22) = 2.54, p < .02$). Thus one reason the novices take longer to debug these programs is that their initial hypotheses were often not correct. This led to more hypotheses and longer debugging sessions.

Another characteristic pattern of behavior that differentiated novices from experts was that novices frequently added bugs to their programs, meaning that they made a change in the code while exploring a hypothesis and then did not undo the change before moving on to another hypothesis. This tendency was strongly related to their likelihood of finding our bug within our time limit. When novices failed to find our bug in the allotted time, 92% of the time they had added at least one bug of their own. When novices did find our bug, they had added a new bug only 23% of the time ($t(17) = 5.46, p < .01$)[2]. In contrast to this novice behavior, only once did an expert ever add a new bug to the program (and interestingly enough, did not find our bug within the time limit).

Running the program is a useful debugging strategy, since it allows one to reason backwards from symptoms in the output to potential problems in the program. Experts ran the program an average of once every 1.9 minutes during their session, while novices did so once every 2.9 minutes ($t(22) = 2.15, p < .05$)[3].

In sum, experts appear to understand the programs better, and as a result generate an accurate hypothesis about what our bug is much more quickly than do novices. Further, the experts almost never complicated their task by adding new bugs. The fact that experts ran the programs more often than novices suggests they might also have more effective troubleshooting strategies. In Experiment 2, we looked to see whether a similar pattern of results emerged when experts and novices debugged a program written in Pascal, a language subjects already knew when they came to the laboratory.

Experiment 2

Method. The subjects were 10 novice programmers who had just finished their first or second course in Pascal and 10 expert programmers who were advanced graduate students in computer science.

All subjects in this experiment knew Pascal. The study was conducted on an IBM PC using the Turbo Pascal programming environment. Since this specific environment was unfamiliar to the subjects, a 25-minute training session was used to familiarize the subjects with the environment and allow them to use its various editing, debugging, and file manipulation tools.

[1]Editing the program to put in test writes did not count as a hypothesis test.

[2]These data were analyzed by calculating a single score for each subject that represented these tendencies. Since each subject debugged three programs, a score of +1 was assigned to those cases where they (a) failed to find our bug and added a new bug or (b) found our bug but did not add any new bug. -1 was assigned to the other two combinations. Thus, to the extent that the scores of individual subjects were significantly greater than 0, the hypothesis that finding our bug and adding new bugs were independent of each other could be rejected. The *t*-score is a test of just this: the extent to which the mean score for the 18 novices exceeded 0.

[3]The *t*-test was performed on the reciprocals of the times in order to normalize the distributions.

Prior to training on the programming environment, each subject read a Pascal program for comprehension. Their understanding of this program was measured by having them (a) answer short questions about the program, and (b) label the functions of various sections of the program. There were two different programs for this phase, with half of each group of subjects given each. This phase of the experiment was part of a transfer design that will not be further described here (since there was no evidence of transfer).

The program to be debugged was a 46-line Pascal program that read in data from a file, manipulated the data, and printed out results. There was a single bug in the program that produced an incorrect value for a counter variable. Subjects were given a listing of the program and a printed copy of the data file, both of which were also available to them in on-line files. They were also given a printed description of the purpose of the program, a copy of the correct output, a calculator, and a notepad. Subjects were allowed 40 minutes to try to get the program to run correctly. They were not told how many bugs there were. Unlike Experiment 1, they were not asked to think out loud. Experts and novices were treated identically in this experiment.

During the debugging session, an observer monitored where the subjects were looking using a second microcomputer located across the room from the subject. The observer recorded on the keyboard where the subject was looking, such as at the screen, the statement of purpose, the program listing, etc. All of the materials were fixed in standard locations in the subject's workspace to simplify the coding process. The observer's keystrokes were time stamped, providing a record of how long a subject looked at each location. For one subject, two observers logged the subject's looking. The two observers agreed on 95.6% of the keystrokes, and all of the disagreements were minor.

Results. Nine of 10 experts found the bug in the allotted time, while only 5 of 10 novices did ($X^2 = 3.81$, $.05 < p < .10$). When the bug was successfully found, experts were much faster than novices. The mean time to solution was 14.2 min. for experts and 33.1 min. for the novices ($t(12) = 5.18, p < .01$).

As in experiment 1, experts and novices differed in their testing of hypotheses, where a hypothesis test was defined as making an explicit editing change in the program and then running it. Experts tested their first hypothesis an average of 12.9 min. into a session, while novices tested their first hypothesis after 24.3 min. ($t(16) = 3.46$, $p < .01$)[4]. Furthermore, experts tested hypotheses of better quality than did novices. Seven of the experts tested the correct hypothesis about what our bug was as their initial hypothesis, while only two novices did ($X^2 = 5.1, p < .025$).

What were experts and novices doing prior to testing their first hypothesis? For each subject we had a complete record of what they were paying attention to. Two of these categories were (a) studying the program (either on the screen or on the printed sheet) and (b) studying the description of the purpose of the program. Experts and novices devoted the same proportion of their time to these activities prior to testing their first hypothesis. Experts spent 58% of their time studying the program and 14% studying the statement of purpose, while novices spent 63% of their time studying the program and 10% studying the statement of purpose. The other categories of behavior (11 in all) were all very low in frequency. Thus, experts and novices did not distribute their activities differently during the time prior to testing their first hypothesis. However, novices took nearly twice as long to get to the point of being ready to test a hypothesis, during which they studied the program and the statement of purpose almost twice as long as the experts (18.5 min. for novices, 11.2 min. for experts, $t(18) = 2.37, p < .05$).

As in the first experiment, novices who failed to find the bug in the allotted time had often added their own bugs. Three of the 5 novices who failed to find our bug also added their own bugs, whereas none of the 5 novices who found our bug added their own bug. No expert ever added a new bug to the program.

Experts ran the program an average of once every 4.0 min., while novices ran it once every 5.8 min ($t = 1.47$, n.s.). This difference was in the same direction as in Experiment 1, but was not significant here.

General Discussion

Three of our findings were quite surprising to us. First was the fact that prior to generating and testing an initial hypothesis, experts and novices distribute their activities in the same way. We had expected that novices might quickly plunge into experimenting with the program,perhaps pursuing long, blind alleys as they edited the program before really understanding it. Instead, novices studied the program intensively before making any changes. They did not do qualitatively different things than the experts. Rather, they just took longer. Second was the finding that despite studying the program much longer, the novices' initial hypotheses were substantially inferior to those of the experts. These were not long or difficult programs, yet despite this, novices seldom found our bug on their first attempt at making a substantive change in the program. Third, we had not at all expected to find that novices make matters even harder for themselves by adding additional bugs in the course of looking for ours. Experts almost never added new bugs to the program.

Why were expert subjects faster and more successful at finding the bug in these simple programs? The overall pattern of behavior of the two groups of subjects was not strikingly different. In experiment 2, both experts and novices waited a long time to test their initial bug hypothesis, usually until more than 70% of their total debugging time was up. And both groups spent most of this time studying the program.

The evidence from these two studies suggests that the primary reason for the experts' superiority was the ease with which they understood what the program does and is supposed to do. This in turn allowed them to isolate the bug more quickly, as indicated by the frequency with which they found the bug on their first attempt. Novices, on the other hand, while spending nearly twice the time in preliminary study of the program and the statement of purpose, did not

[4]One expert and one novice failed to test a single hypothesis (by our definition) during the 40-minute period.

have sufficient programming knowledge to focus in on good hypotheses. Moreover, they frequently made the situation worse by introducing new bugs as they edited the program and tested hypotheses. These added bugs were a major predictor of whether or not the program was ever debugged within the given time limit.

These findings are reminiscent of the research of Chase and Simon (1973) with expert and novice chess players. In both their work and in ours, the most striking difference between the experts and the novices is in the domain knowledge they possess.Experts' superior performance is based more on their ability to quickly and effectively represent the stimulus situation than on their reasoning strategies. McKeithen, Reitman, Rueter and Hirtle (1981) replicated Chase and Simon's findings with expert and novice programmers. In both realms this leads to dramatic differences in problem-solving performance. This is much the same conclusion that Jeffries(1981, 1982) came to in her preliminary studies of debugging.

This study shares a characteristic with most expert-novice studies of problem-solving that has important implications for how we should interpret the results. In order to give experts and novices the same stimuli, rather simple materials are used. As a result, the full range of experts skills are probably not revealed. These studies revealed clear and dramatic differences between experts and novices in the comprehension phase of debugging. But because experts often got the correct hypothesis on their first attempt, the differences in problem-solving behavior following the point at which an initial hypothesis was tested were difficult to study. Many of the problems the novices had after their initial hypothesis seem mainly due to faulty comprehension or retention of the program, as indicated by the extent to which they inadvertently added new bugs. Testing whether experts and novices display differences in troubleshooting strategies probably will require using more difficult programs with the experts.

Acknowledgments

This research was supported in part by grants from the Reading and Learning Skills Center and from the Human-Computer Interaction Laboratory of The University of Michigan. The first author was supported by a traineeship from the National Institute of Mental Health.

References

Chase, W.G., & Simon, H.A. Perception in chess. *Cognitive Psychology*, 1973, *4*, 55-81.

Jeffries, R. Computer program debugging by experts. Paper presented at the meetings of the Psychonomic Society, 1981.

Jeffries, R. A comparison of the debugging behavior of expert and novice programmers. Paper presented at the meetings of the American Educational Research Association, 1982.

McKeithen, K.B., Reitman, J.S., Rueter, H.H.& Hirtle, S.C. Knowledge organization and skill differences in computer programmers. *Cognitive Psychology*, 1981, *13*, 307-325.