Debugging: the Difference between Novices and Experts

Sobhan Badiozamany
sobhan.badiozaman@gmail.com

Haozhu Wang
luffywuliao@gmail.com

ABSTRACT

Debugging is an essential skill for computer science students. It is also a common problem among novice programmers. In this study we compare the debugging strategies and approaches employed by novice programmers and experts using a combination of quantitative and qualitative approaches. The results show the difference between two groups’ approaches. Some practical suggestions to improve programming courses and develop more useful debugging tools are provided.

1. INTRODUCTION AND MOTIVATION

Like many other skills (Fitts 1967), programming have two levels: low level and high level skills. Low level skills are language syntax and simple programming segments, whereas high level skills are knowledge to utilize low level skills in order to construct programming solutions for more complex tasks (Wiedenbeck 1985). In acquiring any new knowledge, there are three phases (Fitts 1967) and (Anderson 1982):

1. “Cognitive phase” in which the learner is instructed. Performance is not smooth at this stage.
2. “Associative stage” in which by training, efficient process is obtained.
3. “Autonomous phase” in which the declarative knowledge is rarely referenced; learner can perform the task automatically.

According to (Gugerty and Olson 1986) when you are given a piece of code written by someone else to debug, the first step is to figure out what the program is doing and what is supposed to achieve (“comprehension phase of debugging”). It also states that debugging skills are improved by practice rather than academic teaching.

According to (Katherine B. McKeithen 1981) and (Shneiderman 1998) the difference between novice and experts is in the different way in which they process information. The best way to start debugging is to find the place in the code where the program starts to behave unexpectedly, and then trace backward to locate the source of problem. Programmers commonly believe that working backward is an effective debugging strategy (Schwartz 1971). This is the concept of program slicing is introduced in (Weiser July 1982 ). The main concept in slicing is that many statements in a given program have no influence on the results of a particular statement. It follows that Slicing is the process of removing statements from a program that have no influence on a particular statement and a variable in that statement. Data flow analysis is one of the automatic techniques of identifying program slices (Hecht 1977).

2. Research Question

Having described debugging concepts in introduction, our research focuses on finding the difference between novice and expert programmers in debugging strategies and skills. Other similar researches (Ahmadzadeh, Elliman and Higgins 2005) (Fitzgerald 2010) (Gugerty and Olson 1986) (Weiser July 1982) (Wiedenbeck 1985) have studied the performance of programmers in debugging code written by someone else. In a different approach in answering this question, our study asks how programmers debug programs written by them. Two aspects of debugging are included in our study, debugging logical problems and syntax problems.

3. Related Literature

Similar to other complex skills, programming skills can be automated. In (Wiedenbeck 1985) it is proven that low level skills are automated in experienced programmers. This automation is very important because it gives the programmer more time to focus on higher level tasks.

In (Wiedenbeck 1985) after checking background information, two groups of programmers, novices and experts, were constructed. Low level programming skills (syntax) and high level skills were examined in this study. All programming constructs were on a basic level, they were written using constructs that...
novice programmers know as well as the experts. In the syntactical experiment, each individual was given a print out of a FORTRAN program to debug. Since introductory programming courses focus on syntactical concepts, even novice programmers are supposed to achieve a good result in this test. But experts turned out to be significantly faster than novices which could only be explained by automation phenomenon. In the second phase, high level skills were examined. Each individual was given a program that had both syntax and semantic problems. Programmer was asked to tell whether or not the program description matched the code. Also in this phase, experts were significantly faster and more accurate than novices. It could again be described by the automation process, experts are faster in low level knowledge, and therefore they can spend more time focusing on high level problem. This results experts’ overall performance to be higher than novices.

Following are suggested by (Wiedenbeck 1985):

1. In order to facilitate automation, repetition in introductory programming tasks may be advantageous.
2. New methods to evaluate programmers can be suggested, for example methods that focus on evaluating automation. (use of automation skills to identify good programmers)
3. It is not useful to provide a programming language that is able to express the same thing in different ways; this slows down the automation process.

(Ahmadzadeh, Elliman and Higgins 2005) Analyzes the patterns of debugging among novice computer students. The research was divided into two phases. In first phase, it investigated the patterns of compiler errors. In the second phase the patterns of logical errors were studied. The compiler errors message data were collected from each subject. It shows that among syntax, semantic and lexical errors, the semantic error take up 63% of the whole. The results are quite interesting. The majority of good debuggers are good programmers, but less than half of good programmers are good debuggers. This implies that a good programmer does not necessarily understand a program written by someone else. This research suggested that in order to improve debugging skills, the study of programming should include more practice of reading other programmers’ code.

In (Gugerty and Olson 1986), two groups of programmers, novice and experts are given the same piece of buggy code written by someone else to debug. Their performance as well as their approach is recorded during the experiment. There were no syntax errors in the programs but some logical errors. The experiment was done in two programming languages, Logo and Pascal. Subjects did not have any Logo knowledge, so first step was to teach each subject Logo programming language and get them to debug simple Logo programs to reinforce their knowledge. The results show that experts can debug more quickly and efficiently than novices. They also reveal that beginners added more bugs to the program instead of debugging the program itself. Another interesting observation was that novices had to try more hypotheses on the program than experts; in fact their early hypotheses were often wrong. One of the wise debugging tactics is running the program. Experts ran the program almost 50% more than novices. It claims that the primary reason for experts performing better is that they were better at “comprehension phase of debugging” as mentioned in the introduction.

(Fitzgerald 2010) Compared the tactics observed during a debugging task and the strategies subjects stated during the follow-up interview. This study suggested that debugging should be clearly taught. Without that, the students might develop some ineffective strategies. It also reported that positive attitude towards bugs and debugging is helpful. Students often get frustrated through debugging without the teachers’ assistance. The teachers’ help is necessary during the debugging of novice programmers. It also suggested that for the novice students without guide, searching the Web for solutions can be a waste of time and should probably be discouraged. Therefore students should be guided how to use online resources.

(Weiser July 1982 ) Reveals that when debugging, expert programmers break the code into coherent pieces that are called slices. Slices are not necessarily contiguous piece of code, one slice could well be scattered over the whole code. Programmers were assigned to debug the buggy program. Afterwards, they were asked if they remember various code fragments (slices), specially the buggy slice. Slices were not necessarily contiguous and variable names were changed. It was observed that if the code is long, this phenomenon is more evident. The study
4. Methodology
Designing a qualitative method to capture all different strategies that programmers use when debugging their programs is not trivial. It is very difficult to design a questionnaire that is neutral and open enough to capture all possible debugging strategies. On the other hand, qualitative methods are often criticized for absence of academic strictness (Golafshani 2003). Therefore, in order to find a more reliable answer to the research question, we decided to use a combination of quantitative and qualitative methodologies. In the quantitative part, we published a survey questionnaire and asked some programmers to answer it. Two groups of programmers were identified based on this questionnaire, novices and experts. Thereafter, we conducted interviews with subjects from each group. As of usual qualitative method (Creswell 2004) we interviewed a smaller set of subjects.

In the survey phase, a pool of 33 subjects from different educational and nationality backgrounds answered our questionnaire. The questionnaire included 4 background information questions followed by three main questions. Here is the background of participants: 27% of them were bachelors, 64% masters, and 9% PhDs. Half of them have studied computer science for 5 years, 36% for 3-5 years, and 12% for 1-2 years. Background questions were later used to separate novices from experts. Background questions were combined into a formula to distinguish between novices and experts. Appendix A contains the questionnaire used in our survey.

In the second phase, we conducted semi-structured interviews (Lindlof 2002) with 8 programmers, 4 novices, and 4 experts. In order to identify novice and experts, we used answers to background question from the survey results to classify each programmer as a novice or an expert. Since questions in the semi-structured interviews were open, it was required that we analyze data from the interviews to make a list of answers to each question. For each question, we did the following: We reviewed all the answers to the question in the first run. In the second run we tried to build a list of answers to the question and calculate the number of interviewees who supported each answer. In cases where an interviewee spoke in favor of more than one answer, we split his/her vote over all favored answers. The outcome of this study was a list of answers per each question; each answer in the list was assigned a number that indicates the number of interviewees who supported it.

Appendix B contains the template we used in our semi structured interviews.

5. Results
5.1 Quantitative Results
The results in table 1 show that experts rarely rely on others for solving syntax problems. This is mainly due to the fact that they have mastered the low level programming skills, also described by skill automation phenomenon in (Wiedenbeck 1985). In other words, experts have managed to automate the low level (syntax related) programming skills. On the other hand, high level skills mentioned in (Wiedenbeck 1985) occupy most of their time. Since automation cannot directly help in high level programming skills, they seek additional help (like friends) in solving logical problem. Furthermore, experts use online resources (like Google and forums) more often; the reason behind this is that most online resources are written in expert’s terms and can be confusing for novices (Fitzgerald 2010).

<table>
<thead>
<tr>
<th>Question/Answers</th>
<th>Novices</th>
<th>Experts</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>What are your choices to solve the syntax problems?</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Checking the language manual</td>
<td>55%</td>
<td>53%</td>
<td>2%</td>
</tr>
<tr>
<td>Browsing online resources</td>
<td>77%</td>
<td>93%</td>
<td>16%</td>
</tr>
<tr>
<td>Asking a friend</td>
<td>66%</td>
<td>40%</td>
<td>26%</td>
</tr>
<tr>
<td>Asking the teacher</td>
<td>33%</td>
<td>0%</td>
<td>33%</td>
</tr>
<tr>
<td>Asking the question in a programming forum</td>
<td>22%</td>
<td>27%</td>
<td>5%</td>
</tr>
<tr>
<td>What are your choices to solve the logic problems?</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Asking a friend</td>
<td>55%</td>
<td>60%</td>
<td>5%</td>
</tr>
<tr>
<td>Asking the teacher</td>
<td>44%</td>
<td>20%</td>
<td>24%</td>
</tr>
<tr>
<td>posting the code in a programming forum and ask for feedback</td>
<td>11%</td>
<td>20%</td>
<td>9%</td>
</tr>
</tbody>
</table>
2010), and second they have not mastered automation process described in (Wiedenbeck 1985).

5.2 Qualitative Results
There is a meaningful difference between the way experts solve compiler errors and novices do. First, novice programmers rarely use compiler error messages in solving compiler errors. The reason, also mentioned in (Ahmadzadeh, Elliman and Higgins 2005) is that to novices compiler error messages could be misleading. They also lack the experience required for interpreting compiler errors properly. Second, as also observed in (Weiser July 1982), experts use slicing in solving complicated compiler errors while novices are unfamiliar with slicing strategy.

<table>
<thead>
<tr>
<th>Question</th>
<th>Answers</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>How do you solve the compiler errors?</td>
<td>Asking friends/teachers</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Searching the internet</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Trying to figure out compiler error message</td>
<td>1</td>
</tr>
<tr>
<td>How do you identify and solve the logical problems?</td>
<td>By tracing line by line (naive debugging)</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>No organized way of debugging, reading the whole code</td>
<td>2</td>
</tr>
<tr>
<td>How do you test your program?</td>
<td>No organized way of testing</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Just test the examples in the specification</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 2. Interview abstract - Novice programmers

Lack of slicing strategies in novices was more evident in solving logical problems, which makes debugging activity very inefficient, especially for long pieces of code. In addition, Experts use more sophisticated debugging tools and techniques than novices.

<table>
<thead>
<tr>
<th>Question</th>
<th>Answers</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>How do you solve the compiler errors?</td>
<td>Syntax errors: just use compiler message, The rest checking internet</td>
<td>2.5</td>
</tr>
<tr>
<td></td>
<td>Slicing the code</td>
<td>1.5</td>
</tr>
<tr>
<td>How do you identify and solve the logical problems?</td>
<td>Slicing the code and running individual tests before integration test.</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>use advanced testing tools and models</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Misunderstood the question</td>
<td>1</td>
</tr>
<tr>
<td>How do you test your program?</td>
<td>Individual test and then whole (integration) test.</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Advanced testing tools and scripts.</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Misunderstood the question</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 3. Interview abstract - Expert programmers

Similar to different debugging strategies mentioned by novices and experts in the interviews, slicing technique can be detected in testing methods employed by experts while it is much less evident in novice testing strategies.

The above results show that novices and experts use different debugging strategies. In contrast, (Wiedenbeck 1985) claims that novices and experts use the same strategies and the only difference is higher accuracy and agility of experts due to the automation.

6. Conclusion
Unlike what is claimed in (Gugerty and Olson 1986), there is a meaningful difference in strategies and skills employed by novices and expert programmers in solving syntactical and logical programming problems. Experts use variety of advanced skills like slicing and automation to improve debugging speed and accuracy. According to (Wiedenbeck 1985) these skills are mostly built by the programmer, rather than being taught in academic formal courses. In order to help novices to enhance their debugging skills, some possible teaching improvements are suggested in the next section.

7. Discussion
The dissimilarity between novices’ and experts’ debugging strategies observed in this study suggests that experts build their “bag of skills” by themselves rather than formal academic training. Since debugging is an essential programming skill, some changes in courses can be suggested. (Fitzgerald 2010) suggests that debugging must be explicitly taught in programming courses. Another teaching related suggestion is that we can teach students to write code in a way that is easier to slice and therefore easier to debug. For instance, encourage them to write relevant pieces of code together.

The automation phenomenon in skill development leads to two suggestions in (Weiser July 1982). First, creating automatic slicing applications that are embedded in IDEs and debugging tools. The proper use of them, however, also requires formal academic courses to include these concepts. Second, in order to assist automation process in novices, repetition of syntactical concepts in beginner programming courses is recommended.

Further studies can investigate how programming courses can explicitly cover debugging strategies and skills.
Appendix A - Survey Questionnaire

1. For how long have you been studying computer science/engineering?
   a. 1-2 years
   b. 3-5 years
   c. more than 5 years

2. What is the academic level of your current studies?
   a. Bachelor
   b. Master
   c. PhD

3. How many programming languages are you familiar with? (intermediate level of programming knowledge)
   a. 1-2
   b. 3-5
   c. More than 5

4. How much time do you spend (on average) to write code in a week?
   a. Rarely
   b. 1-3 hours
   c. 4-7 hours
   d. More than 7 hours

5. What are your choices to solve the syntax problems?
   a. Checking the language manual
   b. Browsing the online source
   c. Asking friends
   d. Asking teachers
   e. Asking the question in a programming forum
   f. Others

6. What are your choices to solve the logic problems?
   a. Asking friends
   b. Asking teachers
   c. Posting the code in a programming forum and ask for feedback
   d. Others

Appendix B - Semi structured interview directions

1. How do you identify and solve the logical problems?
   (Describe logical problems)

3. How do you test your program?

8. References


