Journal of Software Engineering

ISSN 1819-4311

Academic Journals Inc.

www.academicjournals.com

Journal of Software Engineering 6 (1): 1-9, 2012
ISSN 1819-4311 / DOI: 10.3923/jse.2012.1.9

A Study on the Program Comprehension and Debugging Processes of Novice Programmers

[1]Syed Ahmad Aljunid, [2]Abdullah Mohd Zin and [2]Zarina Shukur

[1]Department of Computer Science, Faculty of Information Technology and Computer Science, Universiti Teknologi Mara, Shah Alam, Malaysia

[2]Center For Software Technology and Management, Faculty of Information and Science Technology, Universiti Kebangsaan Malaysia, 43600 Bangi, Malaysia

Corresponding Author: Syed Ahmad Aljunid, Department of Computer Science, Faculty of Information Technology and Computer Science, Universiti Teknologi Mara, Shah Alam, Malaysia

ABSTRACT

This research presents an empirical study on the program comprehension and debugging processes of novice programmers. We provide empirical evidence that the increase exposure to a large number of quality code modification and adaptation in formal teaching is a viable technique for novices to learn program debugging but not for program comprehension. The empirical study is based on case studies at one of the Malaysian universities among the first-degree Information Technology programs students taking Java Programming, an elective programming course. We design a quasi-experiment with non-randomized quota sampling control group with pre-test-post-test. This experiment looks into the program comprehension and debugging constructs at the micro level. Code segments in Java programming language between 5-25 lines of codes are given for the students to try to comprehend or debug manually with pen and paper within a specific timeframe. It will form part of the normal assessment tests for the course. The pre-test involve correct code while the post-test involve both correct and (logical and run-time) bugged code. A control group of 80 students and a treated group of 24 students form the non-randomized quota samples.

Key words: Program understanding, program debugging, empirical software engineering

INTRODUCTION

Generally, program understanding is the process of acquiring knowledge about computer program. Specifically, program understanding is the process of recognizing program plans and extracting design goals in the source code (Quilici *et al*., 1998). Understanding even a small program is a complex task that requires both knowledge and analysis. Indeed, the pattern matching algorithm between plans (represented by schemas, knowledge constraints or plan libraries) and programs (represented by the actual program, annotated abstract syntax trees or flow graphs) has been proven to be NP-hard (Woods and Yang, 1996).

In program comprehension, different research methods have been used and correspondingly diverse tools have been developed. Three main factors have been identified by Storey (2006) for the grounds of this diversity in theories, research methods and tools-the targeted programmers' characteristics, the program characteristics and the software task. Meanwhile, debugging is the task of identifying and correcting faults in code. Once a program failure has been found we must

acquire an understanding of the program so as to localize the program fault and thus identify the program failure. Thus, the twin goals of debugging are to simultaneously develop an understanding of the program and localize the fault area of the code so that adequate correction can be applied (Francel and Rugaber, 1999). However, debugging is an awesome labor-intensive and time-consuming activity. It also requires both analytical and intuitive thinking and this two disparage approaches direct how the programmers debug code (Rosenberg, 1996). It has been reported that testing, analysis and debugging consumes 50% of the cost of developing large software systems (Roongruangsuwan and Daengdej, 2010).

In this research we present an empirical study on the program comprehension and debugging processes of novice programmers. An experiment is carried out in order to investigate the possibility of supplementing their knowledge and skill of program comprehension and debugging via continuous application of appropriate programming pedagogy using lecture, lab sessions, assignments and project work. The purpose of the experiment is to improve the automated program understanding tool that we have developed (Sani *et al.*, 2008, 2009).

The measure of comprehension and debugging will be based on the following criteria. The level of program comprehension required here is not at the physical one-statement level but at the more abstract conceptual block-level. Using Deimel and Naveda (1990) as a reference, the comprehension level referred to in this study is equivalent to the Bloom's taxonomy application level (Level four) and not the understanding level (Level two). Thus, what is meant by comprehension is the students must be able to identify exactly what each code segment is doing or processing and not in terms of lower-level constructs such as how, what identifiers are used, what each statement does which construct or data structure is used, etc.

The general corresponding hypothesis is novices given more quality code to modify and adapt via formal teaching will perform better program comprehension and debugging. This experiment will encompass correct and bugged code segments.

Both the control and the treated groups were given programming assignments which involves code modification and adaptation, as well as code walk-through or code reading sessions in the lab. However, the difference is that the treated group had three extra programming assignments and two extra code reading sessions as compared with the control group. The three extra programming assignments and the two extra medium-size live code reading sessions are based on modifications/adaptations.

In particular, we will address the following hypotheses:

- **H1:** There exist a significant difference in the learning of program comprehension in the treated group as compared to the control group before and after the exposure of a larger number of quality code modification and adaptation

The independent variable in H1, is the larger number of quality code modification and adaptation while the dependent variable is the learning of program comprehension.

- **H2:** There exist a significant difference in the learning of *debugging* in the treated group as compared to the control group before and after the exposure of a larger number of quality code modification and adaptation

Correspondingly, in H2, the independent variable in above is the larger number of quality code modification and adaptation while the dependent variable is the learning of debugging.

MATERIALS AND METHODS

One pilot test each was carried out prior to the experiments as part of the class tests for the course October and April semesters, respectively. A total of 54 and 53 students, respectively were involved. The experiment in this paper is based on the results of these two pilot tests.

The experiment consists of a pre-test and post-test are conducted. Each test will have separate but identical questions for comprehension/debugging for both the control and treated groups. The experiment is based on the partial test scores of the course actual Test 1 and 2, respectively. Both set of test questions are of very similar level and have the same total points (50 marks) and contains the same set of topics. However, the program comprehension and debugging questions for both groups is exactly identical. The pre-test instrument in Test 1 is Question 1 which contains three lines of Java code for the respondents to understand a looping construct applied on string and characters (Fig. 1). The full mark for Question 1 is 5.

The total marks for the two post-test questions, Question 2, 3, 4 and 6, respectively. The questions contain more than 10 lines of Java code. Question 2 involves exception handling and file input-output, whilst Question 3 involves looping construct, exception handling and input validation. Table 1 summarize the experiment.

The setting and participants: Code segments in Java programming language between 5-25 lines of codes are given for the novices to try to comprehend or debug manually with pen and paper within a specific timeframe. It will form part of the normal assessment tests (specifically, Test 1 and 2) for the course. The pre-test involves correct code while the post-test involve both correct and (logical and run-time) bugged code.

The control and instrumented groups consists of 80 and 24 students respectively. However, three students data from the control group have to be dropped because of experimental mortality factor (Key, 1997); the first two did not turn up for the pre-test while the third did not turn up for the post-test. The final tally for the control group is thus 77 subjects only.

The experiment is based on the partial test scores of Test 1 and 2 specific to program understanding and debugging of given Java code segments only. The test is administered and collected during the semester (for the control and treated groups respectively) as part of the overall course assessment.

Table 1: The experiment

	Size	Pre-test	Process	Post-test
Control group	80	Question 1	• Programming assignments (code modification and adaptation) • Live code reading session.	Question 2 and 3
Treated group	24	Question 1	• Extra three programming assignments (code modification and adaptation). • Extra two live code reading session.	Question 2 and 3

Explain, in high level terms, what this code segment really means:

```
int i = 0;
while (i < s.length() &&
Character.isLetter(s.charAt(i)))
i++;
```

Fig. 1: Experiment 1 pre-test question (Q1)

Explain, in high-level terms, what the **bolded** code segment below does:

```
InputStreamReader book = null;

try {
 book = ...
}
catch □ {□ }
finally {
 if (book != null)
 book.close();
}
```

Fig. 2: Experiment 1 post-test question 1(Q2)

Pre-experiment Test: The total mark for the pre-test question, Question 1, is 5.

Question 1 consists of 3 lines of Java code (Fig. 1). It consists of a Java code segment with a single loop with a single increment counter statement. The loop is terminated once any one of the two loop conditions fails. The question seeks to elicit the novices understanding of *what* the code actually does rather than how it is done. As such, the level of understanding required in this question is at the Bloom's analysis level and not merely at the understanding level. Two possible correct answers are:

- The code will skipped over all the alphabets in the string's prefix until a non-alphabet character is found
- It will count the number of non-alphabets found in the string's prefix until a non-alphabet character is found

Post-experiment test: The post-test instrument Test 2 consists of Question 2 and 3; Question 2 contains more than 10 lines of Java code for the respondents to understand on exception handling and file input-output while Question 3 contains more than 10 lines of Java code on the looping construct, exception handling and input validation for the respondents to debug (Fig. 2, 3). The full marks for Question 2, 3, 4 and 6, respectively.

Question 2 consists of a Java code segment highlighting a single if statements inside the try-catch-finally clauses for handling Java file input-output exceptions.

This second question also seeks to elicit the novices understanding of what the code actually does rather than how it is done, i.e. the level of understanding required in this question is at the Bloom's analysis level. The possible correct answer is:

- If the file (denoted by the book variable) has been (successfully) opened, then it will be closed (irrespective of whether there is any exception caught or otherwise)
- Note that the parts in parentheses above are for clarifications and completeness only; a student will score full marks even if he/she does not write these

Question 3 consists of a bugged Java code segment with try-catch clauses for handling Java numerical input validation using a loop.

Question 3 seeks to elicit the novices' knowledge and skill of debugging the code. The cognitive level required in this question is at the Bloom's analysis level. This question also illustrates a very

The code below is supposed to be robust code for entering numerical data. It is supposed to continue looping until the user inputs the correct data format. Otherwise, it will prompt the user with an error message and prompts the user to input again. However, there are several errors in the code. Assumed the Line class has been defined.

```
double base = 0.0;
try {
 while (true)
 base = Integer.parseInt(javax.swing.
 JOptionPane.showInputDialog
 (□Enter the line base-value:□));
}

catch(NumberFormatException e) {
 System.out.println(□Illegal number format! □ +
 e.getMessage());
}

// now the program continues normally
Line c = new Line(base, Math.pow(base,3));
□
```

(a) What are these errors?

(b) Correct these errors.

Fig. 3: Experiment 1 post-test question 2 (Q3)

important type of bug involving novices which is called multiple dependent bugs. Pan (1993) listed it as multiple dependent faults and these types of bugs occur as a result of conglomerated knowledge (Perkins and Martin, 1986). The bugs are a result of situations where the novices produces code that mix-up several disparage elements or constructs into syntactically or semantically ill-formed code.

In Fig. 3, the conglomerated knowledge involves the while loop and the try-catch exception handling constructs. The loop has been wrongly placed inside the try block while simultaneously the loop condition will result in an infinite loop as long as the input is a legal integer value.

To correct these two multiple dependent bugs, a possible correct answer are:

- The while loop encompasses the try-catch blocks and not vice-versa as at present
- The while loop true condition is replaced by a Boolean variable which remains true until a legal numerical data is entered

ANALYSIS AND RESULTS

The data analysis is conducted by using the statistical SPSS tool. The data is first be coded and edited before statistical analysis is done. As the number of respondents is 80 and 24 for the control and treated groups respectively, i.e., one group has more than 30 respondents while the other has less than 30, the non-parametric statistical analysis is used. Non-parametric statistics does not assume the data being tested is a normal distribution; by taking into consideration the benefit of doubt on the normality of each group, especially the second smaller-size group, non-parametric analysis is used. The 2-independent samples Mann-Whitney U test will be performed to determine whether there is a statistically significant difference between the control and treated group median scores. The Mann-Whitney U test is used to test the hypothesis that two independent populations have the same distribution. It is used when the assumption of normality for parametric test is not met and the samples sizes are small (Mann, 2010).

Apart from these, descriptive statistics is used to describe the overall profile of the respondents. These include the frequency distribution and the measures of central tendency (the mean, mode and the median) to summarize the data collected.

The treatment that differentiates both groups is the extra number of quality code modification and adaptation given to the treated group. For the treated group, a total of three extra programming assignments based on modifications/adaptations were given. Each of these is based on different topics and problem domains. Furthermore, the treated group underwent two extra medium-size hive code reading, execution, analysis and modification sessions. These sessions were guided in the lab by the instructor. The code in the first session was taken from an original code downloaded by one of the project groups while the second was an already modified (but flawed) code by another project group.

Thus the treated group had three extra programming assignments and two extra code reading sessions as compared with the control group. Also, to maintain score consistency, one examiner marked all the answers.

In each of the two cases above, code reading and critique was illustrated in the lab for the students to analyze and evaluate on what type of modifications, programming constructs and standardization should be done to make the program correct or more readable, better and/or more efficient. After each of these sessions, an assignment will be given out for the students to modify or adapt. The third assignment, a Java applet game with thread programming and GUI, was given after a live code reading session in the lecture for them to modify.

The first hypothesis, H1, is there a significant difference in the learning of program comprehension in the treated group as compared to the control group before and after the exposure of a larger number of quality code modification and adaptation. The treated group had been given three extra assignments based on modifications and code reading. Accordingly, the second hypothesis is there a significant difference in the learning of debugging in the treated group as compared to the control group before and after the exposure of a larger number of quality code modification and adaptation. As each of the three questions Question 1, 2 and 3 above have different full marks and are based on different topics, a straight comparison cannot be made. Instead, the scores for each question have been standardized to percentage form, using the following formula:

$$\text{Standard\_Score} = \text{Actual\_Score} \times \text{Total\_Marks}/100$$

Once the standard scores have been calculated, the difference between the standard scores is used to measure the learning of program comprehension and debugging constructs for each student based on their both pre-test and post-test results:

$$\text{Difference in comprehension learning} = \text{Standard\_Score\_Question2} - \text{Standard\_Score\_Question1}$$

$$\text{Difference in debugging learning} = \text{Standard\_Score\_Question3} - \text{Standard\_Score\_Question1}$$

These two values are used for the non-parametric Mann-Whitney test statistical analysis to determine whether there is a statistically significant difference between the control and treated group median scores (Table 2, 3).

Table 2: SPSS results of Mann-Whitney Test Ranks

Variables	Group	N	Mean Rank	Sum of Ranks
DiffPostPrePC	Control	77	48.71	3751.00
(Difference between post and pre-tests on program comprehension)	Treated	24	58.33	1400.00
	Total	101		
DiffPostPreDbg	Control	77	47.20	3634.50
(Difference between post and pre-tests on debugging) Treated	24	63.19	1516.50	
	Total	101		

Table 3: SPSS results of Mann-Whitney Test

Test statistics	DiffPostPrePC	DiffPostPreDbg
Mann-whitney U	748.000	631.500
Wilcoxon W	3751.000	3634.500
Z	-1.412	-2.346
Asymp. Sig. (2-tailed)	0.158	0.019

Let π_1 be the median standard score of the control group and π_2 be the median standard score of treated group for the difference in learning program comprehension.

$$H_0 : \pi_1 = \pi_2$$
$$H_1 : \pi_1 \neq \pi_2$$
$$\alpha = 0.05$$

The output of the Mann-Whitney Test indicates that the result Z-score conversion was not significant, z = -1.412. Based on the p-value = .158 is greater than α = .05, we do not reject the null hypothesis at this significant level. Consequently, we conclude there is no significant difference in the learning between the Control and Treated groups for program comprehension (DiffPostPrePC) for hypothesis H1.

Let π_1 be the median standard score of the control group and π_2 be the median standard score of treated group for the difference in learning debugging.

$$H_0 : \pi_1 = \pi_2$$
$$H_1 : \pi_1 \neq \pi_2$$
$$\alpha = 0.05$$

Correspondingly for hypothesis H2, the output of the Mann-Whitney Test indicates that the result Z-score conversion was significant, z = -2.346. Based on the p-value = .019 is less than α = .05, we reject the null hypothesis at this significant level. Consequently, we conclude there is a significant difference in the learning between the Control and Treated groups for debugging (DiffPostPreDbg).

Therefore, hypothesis H1 is rejected but H2 is accepted. The results show that there is no significant relationship between the increase in test scores and learning program comprehension for the treated group. Furthermore, there exist a significant relationship between the increase in test scores and learning debugging for the treated group.

DISCUSSION

The mixed results indicates the learning impact is significant for debugging but not for program comprehension. The negative result for program comprehension can be attributed to the high context nature of the code in Question 2 in Fig. 2. The code segment is produced below:

```
finally {
 if (book != null)
 book.close();
}
```

To answer this question, both the exception handling mechanism in Java, the try-catch block including the optional finally block and Java file input-output mechanism, must be understood. As this kind of good programming style has been taught in the file input-output topic *and* given in file input-output handouts, i.e. to check whether a file has been opened using the optional finally block before proceeding to close it, the students are expected to be able to comprehend this code easily, whether the control or the treated group. Although there is an increase in the score of both groups in Question 2 compared to Question 1, some students may have been put off by file input-output complexity in Java which, compared to their previous programming experience in C++, is much more complicated and cumbersome. The Java designers at Sun Microsystems are well aware of this criticism; Java 5 has now incorporated a simpler input class called Scanner to serve this purpose. However, the Java input-output mechanism is still very complex with a large number of input-output classes which must be used in tandem with the exception handling mechanism of Java.

On the contrary, Question 3 consists of a loop construct for GUI input validation. Although the question also used the exception handling try-catch block, the students' performance suggest that they can comprehend and debug this kind of code. Rather, it is the Java file input-output mechanism which must be implemented simultaneously with exception handing mechanism that they have a hard time in understanding.

Although the pre-test and post-test contains different questions and focus on different topics, this issue has been handled since we do not seek to find and compare the mean scores of each test across both groups using a 2-paired samples statistical testing. We did not investigate the one-on-one the performance of Question 1 vs. Question 2 and Question 1 vs. Question 3 but rather seek to find out whether there exist a significant difference in the learning of treated group as compared to the control group before and after the exposure given above as spelt out by RQ1. However, a better experimental design would be to test the same test questions for the pre-test and post-test before and after the treatment.

REFERENCES

Deimel, L. and J. Naveda, 1990. Reading computer programs: Instructor's guide and exercises. Technical Report CMU/SEI-90-EM-3 ADA228026, Software Engineering Institute, Carnegie Mellon University.

Francel, M. and S. Rugaber, 1999. The relationship of slicing and debugging to program Understanding. Proceeding of the 7th International Workshop on Program Comprehension, May 5-7, 1999, Pittsburgh, PA, USA,.

Key, J.P., 1997. Experimental research and design. http://www.okstate .edu/ag/agedcm4h/academic/aged5980a/5980/newpage2.htm

Mann, P.S., 2010. Introductory Statistics. 7th Edn., John Wiley and Sons, New Jersey.

Pan, H., 1993. Debugging with dynamic instrumentation and test-based information. Ph.D. Thesis, Purdue University, USA.

Perkins, D.N. and F. Martin, 1986. Fragile Knowledge and Neglected Strategies in Novice Programmers. In: Empirical Studies of Programmers, Soloway, E. and S. Iyengar (Eds.). Ablex, New Jersey, USA., pp: 213-229.

Quilici, A., Q. Yang and S. Woods, 1998. Applying plan recognition algorithms to program understanding. J. Automated Software Engin., 5: 347-372.

Roongruangsuwan, S. and J. Daengdej, 2010. A test case prioritization method with practical weight factors. J. Software Eng., 4: 193-214.

Rosenberg, J.B., 1996. How Debuggers Work: Algorithms, Data Structures and Architecture. John Wiley and Sons Ltd., New York., USA., Pages: 256.

Sani, N.F.M., A.M. Zin and S. Idris, 2008. Object-oriented codes representation of program understanding system. Proceeding of the International Symposium on Information Technology, August 26-29, 2008, Kuala Lumpur.

Sani, N.F.Z., A.M. Zin and S. Idris, 2009. Implementation of conceiver: An object-oriented program understanding system. J. Comput. Sci., 5: 1009-1019.

Storey, M.A., 2006. Theories, tools and research methods in program comprehension: Past, present and future. Software Qual. J., 14: 187-208.

Woods, S. and Q. Yang, 1996. Approaching the program understanding problem: analysis and aheuristic solution. Proceedings of the 18th International Conference on Software Engineering, March 25-30, 1996, Berlin, Germany, pp: 589.