

Cognitive Process during Program Debugging

Shaochun Xu, Václav Rajlich
Department of Computer Science
Wayne State University
Detroit, MI 48202
{w20043, vtr}@cs.wayne.edu

Abstract

Program debugging is a critical and complex activity in software engineering. Accurate and fast debugging leads to high quality software and a short time-to-market. Debugging involves a very demanding cognitive process. In a case study, we found all six levels of Bloom's taxonomy of cognitive learning, from "knowledge" through "comprehension", "application", "analysis", "synthesis", and "evaluation". The involvement of the higher levels of Bloom's taxonomy, such as synthesis and evaluation, indicates that program debugging is a difficult cognitive task. This fact may explain the difference between novices and experts in debugging effectiveness.

1. Introduction

Debugging refers to the combined process of testing and code correction. Programmers spend a large part of their effort in debugging [6]. In spite of the progress in the past years, debugging still remains labor-intensive and difficult [1]. It is one of the most challenging areas of software engineering.

Program debugging involves a complex cognitive process and requires comprehension of both program behavior and program structure. Debugging is similar to a diagnosis done by a medical doctor. Programmers are presented the symptoms (error message, exception, or wrong output) and have to find the cause.

Because of time-to-market and quality demands, programmers must be skilled at debugging. Debugging capability depends on the programmer's knowledge, examples of which are program knowledge, domain knowledge, and specific bug knowledge.

Learning is the cognitive process that obtains the required knowledge. The difficulty of learning is classified by the Bloom's taxonomy. Bloom [3] identified three domains of learning: cognitive, affective, and psychomotor. The cognitive domain is for mental skills, the affective domain is for growth in feelings or emotional areas, and the psychomotor domain is for manual or physical skills. Cognitive domain learning has been classified into six learning levels.

In order to understand the cognitive levels used during program debugging, we analyzed a case study of debugging in information system based on COTS (Components Off The Shelf) [10]. We classified the cognitive activities of program debugging into the six levels of Bloom's taxonomy. This approach can help us further understand program debugging and investigate the differences between novices and experts during program debugging in terms of cognitive activity, which in turn can help programmers and their managers improve their debugging capabilities.

Section 2 describes the Bloom's taxonomy. Section 3 describes the process of debugging as a cognitive process. Section 4 contains the case study. Section 5 contains an overview of the related literature. Section 6 contains conclusions and the future work.

2. Bloom's Taxonomy

Beginning in 1948, Bloom headed a group of educational psychologists who undertook the task of classifying educational goals and objectives. Work done on the cognitive domain was completed in 1956 and is commonly referred to as "Bloom's Taxonomy of the Cognitive Domain" [3].

Bloom identified six levels within the cognitive domain, which has at its lowest level the simple recall or recognition of facts, and has increasingly more complex

and abstract mental levels until the levels of synthesis and evaluation [3]. The categories can be considered as degrees of difficulty and proficiency.

Table 1. The cognitive domain taxonomy and illustrative examples

Level	Bloom's Taxonomy	Sample verbs	Sample Behaviors
1	Knowledge	Acquire, identify, recognize, define, name	Student is able to define the six levels of Bloom's taxonomy
2	Comprehension	Explain, describe, interpret, illustrate	Student explains the purpose of Bloom's taxonomy.
3	Application	Apply, relate, use, solve, construct	Student writes an instructional objective for each level of Bloom's taxonomy.
4	Analysis	Analyze, categorize, contrast, discriminate	Student compares and contrasts the cognitive and affective domains.
5	Synthesis	Create, design, specify, propose, develop, invent	Student designs a classification scheme for educational objectives that combines the cognitive, affective, and psychomotor domains.
6	Evaluation	Validate, argue, judge, recommend, justify	Student judges the effectiveness of Bloom's taxonomy.

Table 1 outlines each level of cognition along with illustrating verbs and simple examples that illustrate the type of the thought activity of that level. Bloom's taxonomy is easily understood and is widely used today.

There is some debate in educational circles if the synthesis layer is truly below the evaluation layer [15], but that discussion is beyond the scope of this paper and does not truly impact the conclusions of this paper, as we treat synthesis and evaluation as independent levels.

Bloom's taxonomy has been applied by Buckley and Exton [4] to an assessment of programmers' knowledge.

3. Debugging

A bug is an error in a program that causes the program's behavior to be inconsistent with the programmer's or the user's expectations. It manifests itself during compile-time or run-time.

Debugging is the process of locating and correcting bugs. The errors can be made in various stages of software development or evolution, such as specification, implementation, or debugging of earlier errors [11].

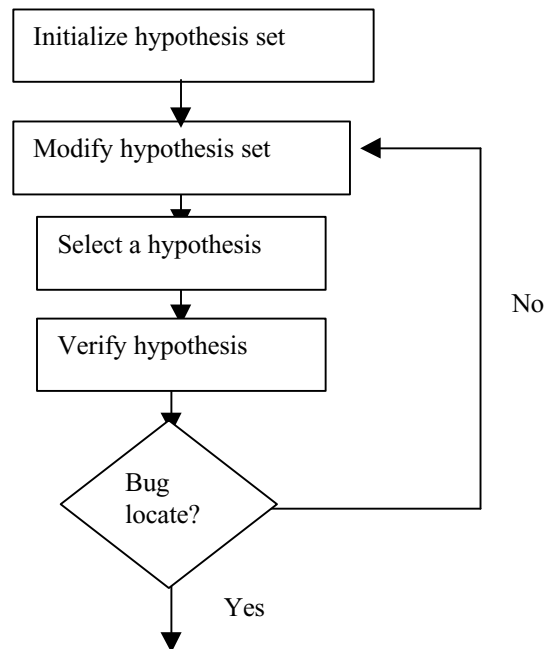


Figure 1. A debugging process model (modified from [1])

Typical activities in the debugging process include program diagnosis, error detection, error removal, and testing [20]. Program diagnosis is analogous to a diagnosis by a doctor. The program displays some "symptoms" and the programmer needs to find the "disease" which causes the symptoms. The programmer

runs and tests the program to obtain appropriate clues, analyzes the symptoms, and uses diagnostic knowledge to generate reasonable hypotheses. The hypotheses are then tested in experiments. The experiments could support or contradict the hypotheses. The process is repeated until the programmer narrows down the bug location by using the isolation strategy (Figure 1). After “treating” the disease (correcting the bug), and testing the program, programmer makes sure that the program is now correct. If not, the process will be repeated until the bug is completely corrected.

In order to make a correct bug diagnosis, programmer must be able to:

- Obtain and understand the meaning of the symptoms of the buggy program
- Identify the appropriate code segments or components involved in the bug
- Identify the process in which the bug is occurring
- Know the most likely cause of the bug
- Evaluate the evidence and select the strategy

Program debugging is a common part of software processes. It is an accepted fact that software engineering processes require a significant amount of time in debugging. The debugging in turn contains program comprehension and testing as the central tasks [5] [14].

4. Case Study

In order to understand the cognitive activities involved in debugging process and their relationship to Bloom’s taxonomy, we conducted a case study¹.

This case study is an observational case study [19]. During this case study, we adapt an earlier case study that deals with system failure in the COTS based Information System. The additional details are in [10].

COTS systems consist of individual components that may be developed by different companies. The probability of failure increases due to integration of individual components. Therefore, debugging a COTS system requires not only the diagnostic skills, but also the knowledge of underlying component technologies.

4. 1. Original Case Study

The case study deals with a system that consists of a COTS Web server and a relational database management system, each provided by a separate vendor. The

application was expected to be used by multiple customers, but a serious problem was reported when it was deployed for the first time. The symptoms of the problem were that the system was either too slow or was not responding at all to user requests. The purpose of the case study was to find and fix the bug.

The programmer was provided with the symptoms “system not responding or too slow”. Based on his experience with the architecture of similar systems, the programmer considered several possible causes. He decided that the most likely cause is the slow database query that slows down access to the web server. The slow web server then blocks access to the site.

Based on this assumption, the programmer used a “browser skeleton” to validate this hypothesis. The browser skeleton is a test harness, in which a program simulates customers connecting to the web server. The average response time was recorded after a browser skeleton was launched. Then the actual web browser was launched to connect to the server and the response time of the server was also recorded, which turned out to be 100 times longer than the response time for the browser skeleton. This test was repeated several times to make sure that the results were stable and repeatable. The test results supported the hypothesis that queries to the database slow down access to the web server. The programmer has narrowed down the cause of the problem, but further work is needed.

The next hypothesis was that the web server could not spawn child processes to handle customer requests. Based on this assumption, another test was conducted using the same test harness to observe the state of all processes. If the hypothesis were true, the number of processes would not increase after a certain period of time. However, the result did show that some new processes were spawned, and the overall number of processes increased. Therefore, the second hypothesis was not supported by the evidence. Nonetheless, a useful observation was obtained: One of the child processes was accumulating CPU time and others were not.

Based on this observation, the second hypothesis was modified as “not all child processes were handling inbound requests”. Therefore, a further investigation was conducted to see what each child was doing during the browser request process. It was found that only one child works properly and other children tried to set a lock on a file that was already locked by another process. Therefore, there was a block or race condition that put other child processes into an infinite loop and blocked their execution.

After reading the Unix manual and going through documentation and configuration settings, the

¹ Formally, the null hypothesis of the case study is: “Bloom taxonomy cannot be applied to the process of debugging”. The case study falsifies this null hypothesis.

programmer noticed that the web server uses system calls `getcontext` and `setcontext`, which are used in user-level threaded applications. The programmer knew that the web server was actually a multi-threaded application with user-level threads and found that the child process was not threading as expected. Therefore a further hypothesis was made that the actual COTS server agent was not multi-threaded and forced the web server to become blocked.

The server agent provides communication between the web server and the relational database management system. The present server agent provided by the database vendor is used with the `www` server-side API although it also supports the Common Gateway Interface (CGI). If the above hypothesis is true, then using the CGI version of the same agent instead of `www` server-side API version of the server agent would be the right solution, as CGI supports multiple threading. A new test was proposed after the `www` server-side API version of the server agent was replaced with CGI version. The test harness and actual web browser were used again. The result demonstrated that the web server was no longer blocked after the replacement. The problem was diagnosed, a replacement was done and a re-test was conducted. Therefore, the debugging was successful and the bug was fixed.

4.2. Applying Bloom's Taxonomy

When we look at the keywords for Bloom's six levels in table 1 and match them against the activities of the original case study, it turns up that all levels of cognition are involved in program debugging process.

Knowledge level cognition is the prerequisite for program debugging. Programmer must be able to recognize the multi-tier architecture of web application, to identify the meaning of the output, and to define the test harness. If programmer does not have this knowledge, it is not possible to perform debugging.

Armed with this knowledge, the programmer can move to the next level, comprehension. The programmer should be able to comprehend how the client-server works, to explain how SQL queries work, to illustrate how the test harness replaces the web server, and interpret the race condition for two or more threads and so forth.

The next level is application. By applying the knowledge of three-tier architecture, the programmer attempts to solve the problem by assuming that it is the slow response to SQL query. The programmer also applies the knowledge of how spawning child processes handle each web request. After the knowledge has been

applied, the programmer analyzes the relationships of different pieces of information. For example, after obtaining the first test result and observing the amount of time used for each response, the programmer contrasts the response time with the actual web browser and the response time with the browser skeleton, and finds the first is 100 times longer.

During debugging, there are several examples of synthesis. The programmer proposes the hypothesis "slow database query slows down the access to the web server", then "the web server can not spawn child process", then later "not all child processes spawned were handling inbound requests" and finally "the COTS server agent was not multi-threaded and forced the web server to block". All these hypotheses are based on a synthesis of all the available information.

On the evaluation level, the programmer validates whether the hypothesis can correctly explain the bug. For example, the programmer validates the hypothesis "the COTS server agent was not multi-threaded and forced the web server to block". In a similar vein, the programmer rejects the second hypothesis "web server could not spawn child processes to handle customer request".

Based on this categorization, the programmer's actions reflect all levels of Bloom's hierarchy, starting with knowledge and comprehension, through application and analysis, and finally synthesis and evaluation. The classification of the activities of the program debugging in terms of Bloom's taxonomy are summarized in Appendix 1.

Table 2. The relationship between the number of activities and the Bloom's cognitive level

Bloom's level	Number of activities
2	5
3	4
4	2
5	4
6	4

Table 2 shows the frequency of activities on each Bloom's cognitive level. Since level 1 is the prerequisite for the debugging process and all its activities, it is not explicitly listed. The activities that correspond to higher levels of Bloom's taxonomy are numerous and indicate that the debugging process is a difficult cognitive task.

As a result of the case study, we can reject the null hypothesis and state that at least in some instances, Bloom's hierarchy interprets the cognitive process related to debugging well.

5. Related Work

Bloom's taxonomy has been studied widely in education. Teachers often use it to guide their instruction and to prepare questions for students [15].

Vessey and Gould [9], [17] conducted research in the cognitive aspects of debugging and characterized debugging as an iterative process of synthesizing, testing, and refining hypotheses about fault locations and repairs. Yoon and Garcia [20] analyzed the debugging process and investigated how tools can help programmers to obtain the clues about the bugs. Araki [1] provided a general discussion on the framework of debugging and emphasized the process of developing hypotheses.

In [20], two debugging techniques were identified: comprehension strategy and isolation strategy. In comprehension strategy, the programmer searches for the difference between the program requirements and the programs' actual behavior. In isolation strategy, the programmer makes hypotheses about the bugs and then searches for bug locations in the program. Model-based debugging techniques have been studied by [8]. Mateis et al. [13] described a technique to diagnose errors involving objects and reference aliasing.

Three steps of debugging process have been recognized by Uchina et al. [16]: program reading, bug location and bug correction. Ko and Myers [11] have presented a model of programming errors.

Concept location constructs the mapping between programming concepts and the related software components [2]. Once the cause of the bug has been determined, concept location locates the components that implement the concept associated with the bug. Wilde et al. [18] developed a concept location technique called Software Reconnaissance that is based on analysis of program execution traces. Chen and Rajlich [7] proposed a method based on a search of static code that is represented by Abstract System Dependence Graph and consists of control flow and data flow dependencies among software components.

6. Conclusions and Future Work

In this paper, we studied the cognitive aspect of program debugging and applied Bloom's taxonomy to it. The activities in program debugging were classified by different levels in Bloom's hierarchy. The case study showed that for debugging, programmers have to utilize

all six levels of Bloom's hierarchy in order to be able to make the necessary updates.

We speculate that novices may spend more time in accumulating knowledge and in comprehending. In other words, novices likely spend more time on the two lowest levels of the hierarchy. Experts already possess this knowledge and comprehension and their effort concentrates on application, analysis, synthesis and evaluation. We believe that this is the cause of the large gap in the performance of novices and experts. This coincides with the work of von Mayrhauser and Vans [12], who found that programmers who have more domain knowledge would perform program comprehension at a higher-level abstraction. In terms of our case study, a novice might have trouble identifying the cause of the slow response to SQL query whereas an expert immediately recognizes it as the problem of creating child process. Once novice programmers accumulate more knowledge, they move more easily to higher levels such as analysis and synthesis. .

Classifying program debugging activities by the levels of Bloom's taxonomy can help us to better understand the cognitive activities involved. It can shorten the debugging learning process and explain the difference between the novice and the expert.

Future work includes additional case studies of debugging and other software engineering processes using Bloom's taxonomy. These studies will help us to better understand cognitive activities and the structure of the programming knowledge. An experiment on expert and novice programmers in the debugging process will be conducted to study the differences of cognitive activities involved.

References

- [1] Araki, K., Furukawa, Z., and Cheng, J., "A general framework for debugging", *IEEE Software*, 1991, May, pp. 14-20.
- [2] Biggerstaff, T., Mitbender, B., and Webster, D., "Program understanding and the concept assignment problem", *Communications of the ACM*, 1994, May, 37 (5), pp. 72-83.
- [3] Bloom, B. S. (ed.), *Taxonomy of educational objectives: The classification of educational goals: Handbook, I, cognitive domain*, New York, Toronto, Longmans, Green, 1956.
- [4] Buckley, J. and Exton, C., "Bloom's taxonomy: A framework for assessing programmers' knowledge of software systems", *Proceedings of 2nd International*

- Conference on Cognitive Informatics*, 2003, pp. 165-174.
- [5] Brook, R., "Towards a theory of comprehension of computer programs", *International Journal of Man-Machine Studies*, 1983, 18, pp. 86-98.
- [6] Brooks, F., *The Mythical Man-Month*, Addison-Wesley, MA, 1975.
- [7] Chen, K. and Rajlich, V., "Case study of feature location using dependence graph", *Proceedings of IEEE International Workshop on Program Comprehension*, Los Alamitos, CA, 2000, pp. 241-249.
- [8] Ducasee, M., "A pragmatic survey of automatic debugging", *Proceedings of the 1st International workshop on automated and algorithmic debugging*, 1993, Springer LNCS 749, pp. 1-15.
- [9] Gould, J. D., "Some psychological evidence on how people debug computer programs", *International Journal of Man-machine Studies*, 1975, 7(1), pp. 151-182.
- [10] Hissam, S., *Case study: Correcting system failure in a COTS information system, SET monographs on the use of commercial software in government systems*. Technical report, Carnegie Mellon University, 1997, pp. 1-12.
- [11] Ko, A. J. and Myers, B. A., "Development and evaluation of a model of programming errors", *IEEE Symposia on Human-Centric Computing Languages and Environments*, Auckland, New Zealand, 2003, October, pp. 7-14.
- [12] Von Mayrhauser, A. and Vans, A. M., "Program understanding behavior during debugging of large scale software", *Empirical Studies of Programmers*, 7th workshop, Washington, DC, 1986, pp. 213-229.
- [13] Mateis, C., Stumptner, M., Wieland, D., and Wotawa, F., "Model-based debugging of Java programs", *Proceedings of the 4th International Workshop of Automated and Algorithmic Debugging*, 2000, pp. 32-40.
- [14] Shneiderman, B. and Mayer, R., "Syntactic/Semantic interactions in programmer behavior: A model and experimental result", *International Journal of Computer and Information Sciences*, 1979, 8, pp. 219-238.
- [15] Tiene, D. and Ingram, A., *Exploring current issue in educational technology*, New York, NY, McGraw-Hill, 2001.
- [16] Uchida, S., Monden, A., Iida, H., Matsumoto, K., Inoue, K. and Kudo, H., "Debugging process models based on changes in impressions of software modules", *Proceedings of International Symposium on Future Software Technology*, 2000, Aug., Guiyang, China, pp. 57-62.
- [17] Vessey, I., "Expertise in debugging computer programs: A process analysis", *International Journal of Man-Machine Studies*, 1985, 23(5), pp. 459-494.
- [18] Wilde, N., and Scully, M., "Software reconnaissance: Mapping program features to code", *Software Maintenance: Research and Practice*, 1995, 7, pp. 49-62.
- [19] Yin, R. K., *Case Study Research: Design and Methods*. Vol. 5. Applied Social Research Methods Series. Sage Publications Inc. Thousand Oaks, CA, 1994.
- [20] Yoon, B. and Garcia, O. N., "Cognitive activities and support in debugging", *Proceedings of 4th Annual Symposia on Human Interaction with Complex System*, 1998, pp. 160-169.

Appendix 1. Cognitive levels of the debugging activities

No	Activity	Bloom's level
1	The programmer was provided with the symptoms “system not responding or too slow”. Based on his experience with the architecture of similar systems, the programmer considered several possible causes. He decided that the most likely cause is the slow database query that slows down access to the web server. The slow web server then blocks access to the site.	5
2	Based on this assumption, the programmer used a “browser skeleton” to validate this hypothesis. The browser skeleton is a test harness, in which a program simulates customers connecting to the web server.	3
3	The average response time was recorded after a browser skeleton was launched. Then the actual web browser was launched to connect to the server and the response time of the server was also recorded.	2
4	The result turned out to be 100 times longer than the response time for the browser skeleton after contrasting the two response times. This test was repeated several times to make sure that the results were stable and repeatable.	4
5	The test results supported the hypothesis that queries to the database slow down access to the web server. The programmer has narrowed down the cause of the problem, but further work is needed.	6
6	The next hypothesis was that the web server could not spawn child processes to handle customer requests.	5
7	Based on this assumption, another test was conducted using the same test harness to observe the state of all processes. If the hypothesis were true, the number of processes would not increase after a certain period of time.	3
8	However, the result did show that some new processes were spawned, and the overall number of processes increased. Therefore, the second hypothesis was not supported by the evidence.	6
9	Nonetheless, a useful observation was obtained: One of the child processes was accumulating CPU time and others were not.	2
10	Based on this observation, the second hypothesis was modified as “not all child processes were handling inbound requests”	5
11	A further investigation was conducted to see what each child was doing during the browser request process. It was found that only one child works properly and other children tried to set a lock on a file that was already locked by another process.	4
12	Therefore, there was a block or race condition that put other child processes into an infinite loop and blocked their execution.	6
13	After reading the Unix manual and going through documentation and configuration settings, the programmer noticed that the web server uses system calls getcontext and setcontext, which are used in user-level threaded applications.	2
14	The programmer knew that the web server was actually a multi-threaded application with user-level threads and found that the child process was not threading as expected.	3
15	Therefore a further hypothesis was made that the actual COTS server agent was not multi-threaded and forced the web server to become blocked.	5
16	The server agent provides communication between the web server and the relational database management system. The present server agent provided by the database vendor is used with the www server-side API although it also supports the Common Gateway Interface (CGI).	2
17	If the above hypothesis is true, then using the CGI version of the same agent instead of www server-side API version of the server agent would be the right solution, as CGI supports multiple threading.	6
18	A new test was proposed after the www server-side API version of the server agent was replaced with CGI version. The test harness and actual web browser were used again.	3
19	The result demonstrated that the web server was no longer blocked after the replacement.	2