Volume 5, Issue 6, June 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Model-Based Technique Used Debugging Tool

Mahendran Singaraj*, Anil Kumar Mishra, Rakesh Krishnan
Ashoka Institute of Engineering & Tech
India

***Abstract*— The idea is to developing a debugging tool for finding errors and rectifying those particular errors and displaying warning messages. This developing tool to support the software engineer in locating bugs in programs has been an active research area. The task of Model based technique used debugging tool by summarizing the different approaches to model-based debugging that have been proposed in the literature up to the current state of the art presented by the abstraction-based model.**

***Keywords*— Model Based Debugging, IDE, Multiple platform, code debugging**

I. INTRODUCTION

Here we focus on imperative and object-oriented languages[1][3]. This tool to support the program developers, for locating bugs in programs. Increasingly, complex programs require more and more effort to comprehend and maintain them. Several different approaches have been developed, using syntactic and semantic properties of programs and languages. Nowadays, many different techniques and tools have been proposed to attack the software maintenance problem, in particular the debugging problem[5][6]. The focus has largely been on addressing the technical complexity of this task by reducing the size of programs under consideration using heuristics, controlled experimentation and pruning techniques, and on helping developers to control and investigate program executions. However, the intrinsically difficult task to create, maintain and confirm suitable fault hypotheses based on observed correct and faulty program executions has remained largely the user's responsibility. To reduce the cognitive load when debugging requires capturing semantics and intent of program fragments and their interaction. Using different "models" reflecting different aspects of a program under consideration, it becomes possible to support users in managing fault hypotheses in a way that is aligned closely with their view of the program[10]. Here, model-based debugging techniques aim to fill the gap by providing a comprehensive framework that relieves a user from manually matching the observed program execution(s) to ours intuition of how the program should behave. By using models as abstractions facilitating the matching of symptoms to expected behavior, intelligent debugging tools can be devised that assist users in controlling experiments to discriminate between different fault possibilities

MODEL BASED TEST CASE COLLECTION

In this model based test case collection module we want to enter the error making syntax and test cases next we upload those particular details and the warning messages[9][10]. Here, model-based debugging techniques aim to fill the gap by providing a comprehensive framework that relieves a user from manually matching the observed program execution(s) to his intuition of how the program should behave[2][3]. By using models as abstractions facilitating the matching of symptoms to expected behaviors, intelligent debugging tools can be devised that assist users in controlling experiments to discriminate between different fault possibilities[4]. In the following, the ideas underlying the model-based debugging paradigm are outlined.

II. PROBLEM STATEMENT

An easy way to comply with the conference paper formatting requirements is to use this document as a template and simply type your text into it. In recent decades, many different techniques and tools have been proposed to attack the software maintenance problem, in particular the debugging problem. We believe that the general debugging problem can only be solved by combination of different approaches. In this context, the Model Based Software Debugging approach has potential, with the MBSD engine as central component, unifying different debugging approaches through the common abstraction into components and fault modes, and its suitability for an iterative, interactive debugging session that reduces the number of diagnoses through incremental (abstract) specification of test cases.

III. PROPOSED SYSTEM

A. Reducing the size of programs under consideration using discover, controlled experimentation and pruning techniques.

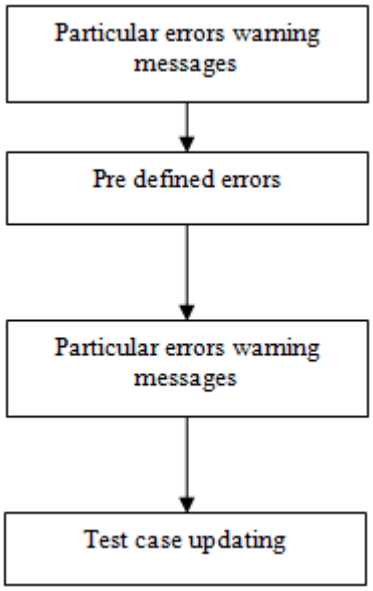
B. To reduce the knowing load when debugging requires capturing semantics and intent of program fragments and their interaction.

C. Here we are providing a simple user interaction for incremental specification of complex program behaviours.

D. Ultimately, extensions to existing approaches capable of isolating complex faults and missing code are desired to cope with general programs.

1. MODEL BASED TEST CASE COLLECTION

In this model based test case collection module we want to enter the error making syntax and test cases next we upload those particular details and the warning messages. Here, model-based debugging techniques aim to fill the gap by providing a comprehensive framework that relieves a user from manually matching the observed program execution(s) to his intuition of how the program should behave. By using models as abstractions facilitating the matching of symptoms to expected behaviours, intelligent debugging tools can be devised that assist users in controlling experiments to discriminate between different fault possibilities. In the following, the ideas underlying the model-based debugging paradigm are outlined.

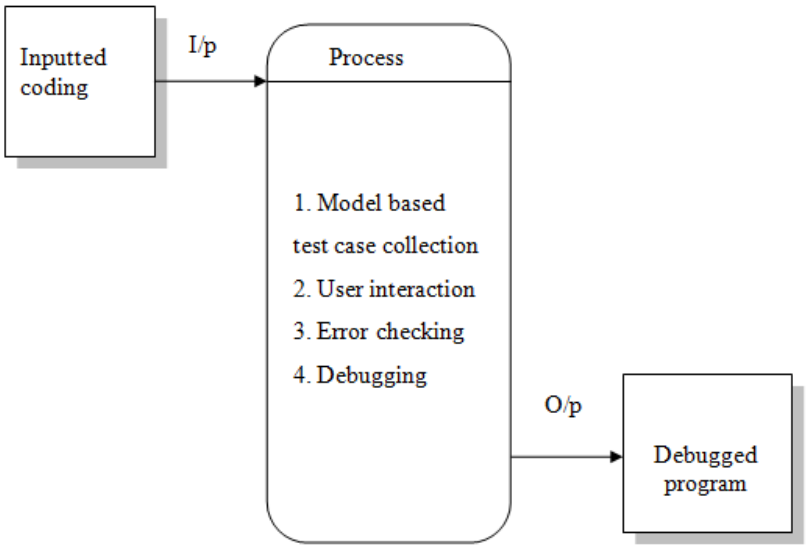


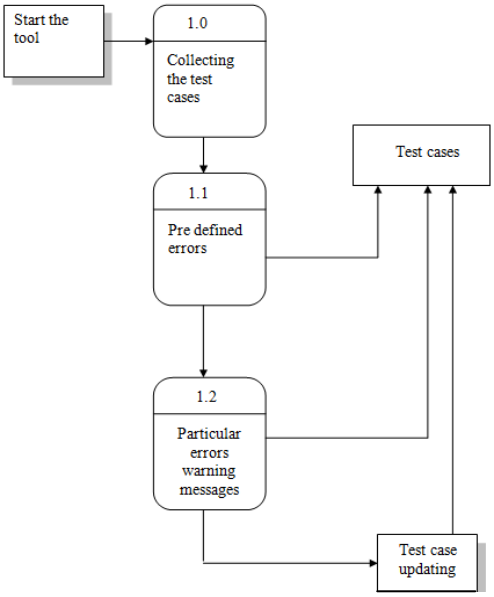
2. USER INTERACTION MODULE

Here the user interacts with the debugging tool for entering model, complex program or upload any programs errors bugging. You enter the programs in the corresponding text area. And check the code with in the use of debugging tool. A set of test programs has been used to evaluate the performance of different debugging approaches. The program was taken from the particular text area, a test case commonly used in the debugging community, and transcribed to Java. The remaining programs have been retained from earlier tests of the dependency of the existing project. For each program, *n* variants with different faults were created by seeding single and double faults, and programs were tested under different test specifications.

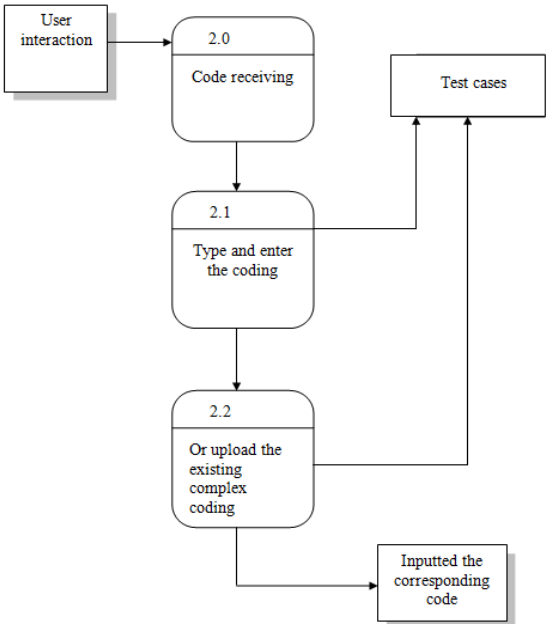
LEVEL BASED DATA FLOW DIAGRAM

Context Flow Diagram (0- level) DFD

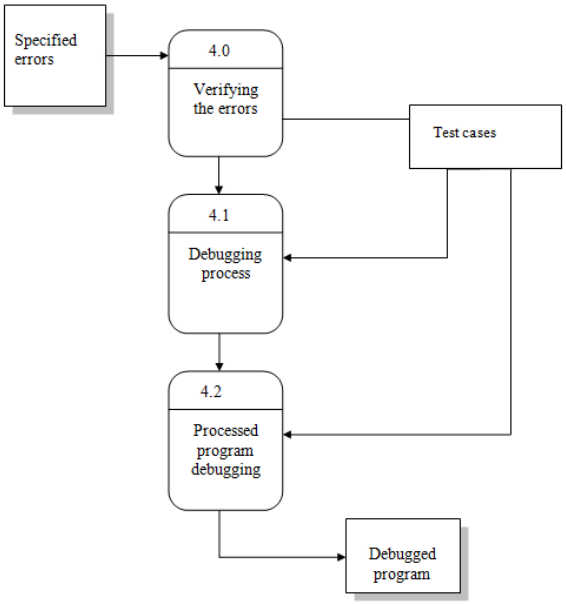




1-LEVEL DFD FOR USER INTERACTION

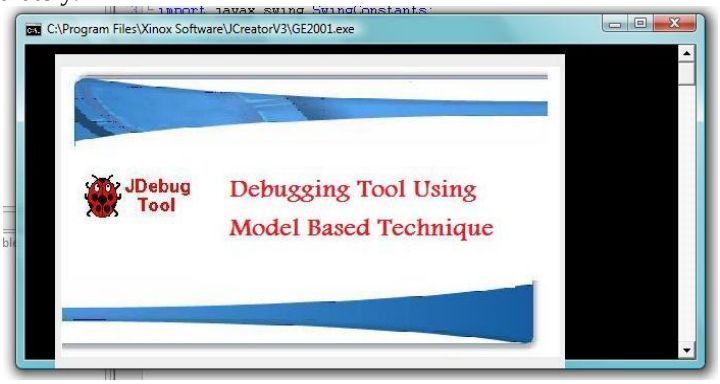


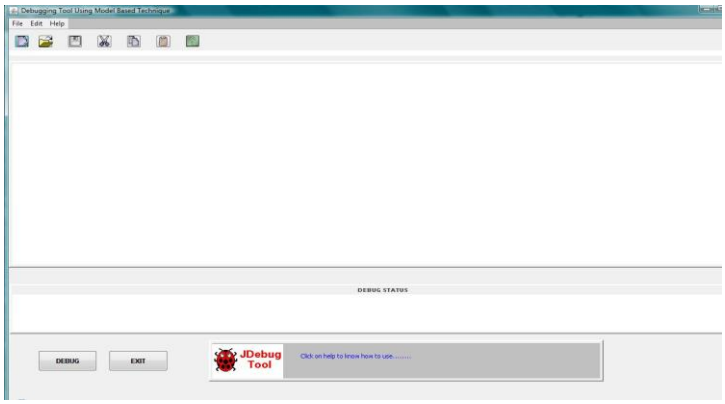
1-LEVEL DFD FOR DEBUGGING

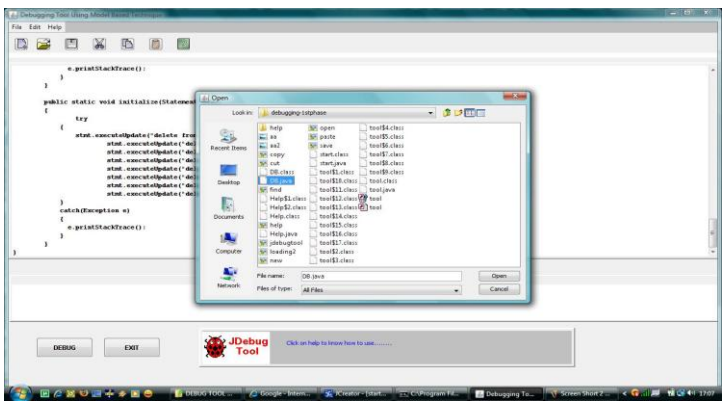


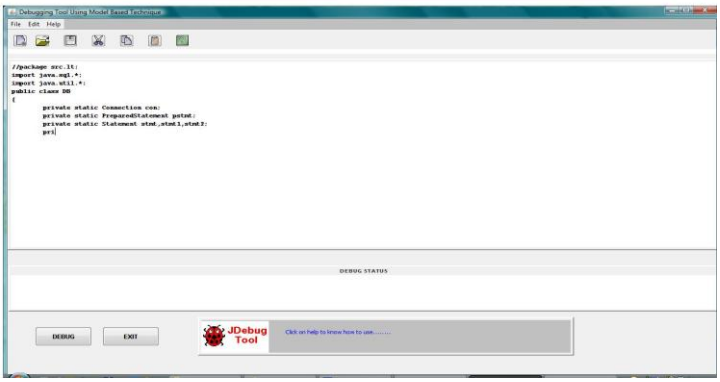
IV. EXPERIMENTAL RESULT

The experimental results are obtained by implementing the Java Applet code. The debugging module in this experiment is giving the result as shown in the screenshot given below. The key idea of adapting Model Based Debugging for debugging is to exchange the roles of model and actual system: the model reflects the behaviour of the (incorrect) program, while test cases specify the correct result. Differences between the observed program execution and the result anticipated by test cases are used to compute model elements that, when assumed to behave differently, explain an observed misbehaviour. The program's instructions are partitioned into a set of model components which form the building blocks of explanations. Each component corresponds to a fragment of the program's source text, and diagnoses can be expressed in terms of the original program to indicate a potential fault to the programmer. This tool is used to easily rectify the errors accurately.









V. CONCLUSIONS

Here we conclude this project has provided an overview of the state of the art in model-based software debugging representations: model-based diagnosis techniques are used to localize faults, starting out from the source code and test cases without an extra formal specification. We provided the first multi-technique empirical comparison of source code fault localization approaches. We have illustrated the relationship between dependency-based approaches that can be applied to large programs but can lead to coarse results to execution based and abstract frameworks. Here to developing this debugging tool for finding errors and rectifying those particular errors and displaying warning messages. This project is used to reduce the cognitive load when debugging requires capturing semantics and intent of program fragments and their interaction.

REFERENCES

[1] H. Cleve and A. Zeller, *Locating causes of program failures,* in Proceedings of the International Conference on Software Engineering (ICSE), G.-C. Roman, W. G. Griswold, and B. Nuseibeh, Eds. ACM Press, 2005, pp. 342–351.

[2] H. He and N. Gupta, *Automated debugging using path-based weakest preconditions,* in Proceedings of the International Conference on Fundamental Approaches to Software Engineering (FASE), ser. Lecture Notes in Computer Science, M. Wermelinger and T. Margaria, Eds., vol. 2984. Springer-Verlag, 2004, pp. 267–280.

[3] D. Wieland, *Model-Based Debugging of Java Programs Using Dependencies,* Ph.D. dissertation, Technische Universit¨at Wien, Nov. 2001.

[4] W. Mayer and M. Stumptner, *Model-based debugging using multiple abstract models,* in Proceedings of the 5th International Workshop on Automated and Algorithmic Debugging, AADEBUG '03, Ghent, Sep. 2003, pp. 55–70.

[5] D. K¨ob, R. Chen, and F. Wotawa, *Abstract model refinement for model based program debugging,* in Proceedings of the Sixteenth International Workshop on Principles of Diagnosis, Monterey, California, USA, Jun. 2005, pp. 7–12.

[6] Wolfgang Mayer and Markus Stumptner, Abstract Interpretation of Programs for Model-Based Debugging, University of South Australia, pp.471-476

[7] Cristinel Mateis, Markus Stumptner, and Franz Wotawa. A Value-Based Diagnosis Model for Java Programs. In Proc. DX'00 Workshop, 2000

[8] Franc¸ois Bourdoncle. Abstract debugging of higherorder imperative languages. In Proc. SIGPLAN Conf. PLDI, pages 46–55, 1993.

[9] Wolfgang Mayer and Markus Stumptner. Model-based debugging – state of the art and future challenges. In Workshop on Verification and Debugging, 2006.

[10] Franz Wotawa. On the Relationship between ModelBased Debugging and Program Slicing. Artificial Intelligence, 135(1–2):124–143, 2002.'

