AUTOMATED TESTING TOOLS FOR OBJECT ORIENTED SOFTWARES

[1]Atul Pratap Singh, [2]Amit Kishor, [3]Pushpendra Kumar Dwivedi

[1,2,3]Department of Computer Science, IIMT Engineering College, Meerut (India)

ABSTRACT

By an automated testing tool, we mean a tool that automates a part of the testing process. It can include one or more of the following processes: test strategy generation, test case generation, test case execution, test data generation, reporting and logging results. This paper deals with design and development of an automated testing tool for Object Oriented Software. By object-oriented software we mean software designed using OO approach and implemented using a OO language.

Testing of OO software is different from testing software created using procedural languages. Several new challenges are posed. In the past most of the methods for testing OO software were just simple extensions of existing methods for conventional software. However, they have been shown to be not very appropriate.

This paper has mainly focused on testing design specifications for OO software. As described later, there is a lack of specification-based testing tools for OO software. An advantage of testing software specifications as compared to program code is that specifications are generally correct whereas code is flawed. Moreover, with software engineering principles firmly established in the industry, most of the software developed nowadays follows all the steps of Software Development Life Cycle (SDLC).

Testing is conducted at 3 levels: Unit, Integration and System. At the system level there is no difference between the testing techniques used for OO software and other software created using a procedural language, and hence, conventional techniques can be used. This tool provides features for testing at Unit (Class) level as well as Integration level. Further a maintenance-level component has also been incorporated. Results of applying this tool to sample Rational Rose files have been incorporated, and have been found to be satisfactory.

Keywords: Class, Object, SDLC, Object-oriented, Testing, Unit, Integration, System, UML, Control flow graph, State transition diagram, Design, Testing, Analysis, Implementation, Black-Box, White-box.

I INTRODUCTION

Software testing is a phase of SDLC that entails much effort, time and cost. Often, testing phase is the single largest contributor towards the whole development time. Testing can not only uncover bugs in the program, but also flaws in design of the software. To make the testing phase quicker, easier and more efficient, automated testing tools are being used. These tools help in test case generation, reporting results and variance from expected ones (if any), bugs in code and other flaws. Usage of these tools speeds up the testing process and also ensures reduction in the probability of a bug/error being uncovered later. However application of these automated testing tools in software testing has its own disadvantages, namely, learning the tool to use it, adapting it to your purpose, and also the tool may not provide specific functionality which you may desire.

Object-oriented testing essentially means testing software developed using object-oriented methodology.

The target users for the Testing Tool are mainly software testers and maintainers. As the tools would provide valuable insight into the program's structure and behavior plus automate the testing process to a certain extent, it would be highly useful for testers. Also the tool would be beneficial to maintainers who would like to study change impact (here they will be aided by the program's analysis done by the tool), and perform regression testing. The objectives of developing the Testing Tool for software testers and maintainers are:

(1) To help them understand the structures of, and relations between, the components of an OO program.

(2) To give them a systematic method and guidance to perform OO testing and maintenance and also suitable due to their effective applicability to OO programs.

(3) To facilitate them to prepare test cases and test scenarios.

(4) To generate test data and to aid them in setting up test harnesses to test specific components.

II OBJECTIVE

The objective of this paper is: design and development of an automated testing tool for object-oriented software. The aim of this paper is to study various established as well as emerging testing techniques, with special focus on the object oriented softwares.

III EXISTING TESTING TECHNIQUES SURVEYED

3.1 Black Box Testing

It is also known as functional testing. A software testing technique whereby the internal workings of the item being tested are not known by the tester. For example, in a black box test on a software design the tester only knows the inputs and what the expected outcomes should be and not how the program arrives at those outputs. The tester does not ever examine the programming code and does not need any further knowledge of the program other than its specifications.

3.1.1 Black Box Testing Advantages

- The test is unbiased because the designer and the tester are independent of each other.
- The tester does not need knowledge of any specific programming languages.

- The test is done from the point of view of the user, not the designer.
- Test cases can be designed as soon as the specifications are complete.

3.1.2 Black Box Testing Disadvantages

- The test can be redundant if the software designer has already run a test case.
- The test cases are difficult to design.
- Testing every possible input stream is unrealistic because it would take a inordinate amount of time; therefore, many program paths will go untested

3.2 White Box Testing

White Box Testing (also known as Clear Box Testing, Open Box Testing, Glass Box Testing, Transparent Box Testing, Code-Based Testing or Structural Testing) is a software testing method in which the internal structure/design/implementation of the item being tested is known to the tester. The tester chooses inputs to exercise paths through the code and determines the appropriate outputs. Programming know-how and the implementation knowledge is essential. White box testing is testing beyond the user interface and into the nitty-gritty of a system.This method is named so because the software program, in the eyes of the tester, is like a white/transparent box; inside which one clearly sees.White Box Testing is contrasted with Black Box Testing. View Differences between Black Box Testing and White Box Testing.

3.2.1 White Box Testing Advantages

- Testing can be commenced at an earlier stage. One need not wait for the GUI to be available.
- Testing is more thorough, with the possibility of covering most paths.

3.2.2 White Box Testing Disadvantages

- Since tests can be very complex, highly skilled resources are required, with thorough knowledge of programming and implementation.
- Test script maintenance can be a burden if the implementation changes too frequently.
- Since this method of testing it closely tied with the application being testing, tools to cater to every kind of implementation/platform may not be readily available.
- White Box Testing is like the work of a mechanic who examines the engine to see why the car is not moving.

IV TESTING TECHNIQUES FOR OBJECT-ORIENTED SOFTWARES

Certain subset of the testing techniques covered in the study can be favorably applied to object-oriented programs. At various levels of testing of object oriented software, techniques which can be applied are [Pressman, iv]:

1. Unit Testing
2. Method Testing
3. Class Testing
4. Integration Testing
5. System Testing

4.1 Challenges to Testing Object-Oriented Systems

A main problem with testing object-oriented systems is that standard testing methodologies may not be useful. Smith and Robson [7] say that current IEEE testing definitions and guidelines cannot be applied blindly to OO testing, because they follow the Von Neuman model of processing. This model describes a passive store with active processor acting upon the store. It requires that there be an oracle to determine whether or not the program has functioned as required, with comparison of performance against a defined specification." They also present the following definition of the testing process: "The process of exercising the routines provided by an object with the goal of uncovering errors in the implementation of the routines or the state of the object or both."Smith and Robson say that the process of testing OO software is more difficult than the traditional approach, since programs are not executed in a sequential manner. OO components can be combined in an arbitrary order; thus defining test cases becomes a search for the order of routines that will cause an error.
Siepmann and Newton[8] agree that the state-based nature of OO systems can have a negative effect on testing. Siepmann and Newton state that the iterative nature of developing OO systems requires regression testing between iterations. Smith and Robson state that inheritance is problematic; since the only way to test a subclass is to flatten it by collapsing the inheritance structure until it appears to be a single class. When this is done, the testing effort for the super class is not utilized; therefore, duplicated testing takes place.

4.2 Study of Testing Techniques For Object-Oriented Systems

Most research on object-oriented (OO) paradigms has been focused on analysis, design, and programming fundamentals. Testing the systems that are created with these paradigms has been considered an afterthought. Traditional testing techniques must be evaluated to determine if they are still useful with respect to object- oriented systems, and new techniques must be developed.

4.3 Latest Research

The latest research in the field of object-oriented software testing. Tonella [20] proposes a method for evolutionary testing of classes. In this paper, a genetic algorithm is exploited to automatically produce test cases for the unit testing of classes in a generic usage scenario. As , object oriented programming promotes reuse of classes in multiple contexts, the unit testing of classes cannot make too strict assumptions on the actual method invocation sequences, since these vary from application to application. Traore [21] discusses a test model for object-oriented programs, based on formal specifications like UML, built from user requirements. Pezze & Young [22] have highlighted some important issues to be considered while testing object-oriented programs. Object oriented software requires reconsidering and adapting approaches to software test and analysis.

V THE TEST MODEL AND ITS CAPABILITIES

The tools for automated testing are based upon certain models of software/programs and algorithms. This mathematically defined test model consists of following types of diagrams:

1. The class diagram (object relation diagram)
2. The control flow graph (of a method), and
3. The state transition diagram (of a class)

5. 1 Class Diagram

A class diagram or an object relation diagram (ORD) represents the relationships between the various classes and its type. Types of relationships are mainly: inheritance, aggregation, and association. In object oriented programs there are three different relationships between classes. They are inheritance, aggregation and association.

5.2 Control Flow Graph

A control flow graph represents the control structure of a member function and its interface to other member functions so that a tester will know which at is used and/or updated and which other functions are invoked by the member function.

5.3 State Transition Diagram

A STD or an Object State Diagram (OSD) represents the state behavior of an object class. Now the state of a class is embodied in its member variables which are shared among its methods. The OSD shows the various states of a class (various member variable values), and transitions between them (method invocations).

5.4 Based On Software Design/Specification

These diagrams are taken from the design models prepared as part of Software Development Process. UML (Unified Modeling Language) has become the defacto standard for object-oriented analysis and design (OOAD). UML provides features for specifying all the above types of diagrams. Rational Rose Suite is the most widely used.

VI COMPONENTS OF THE OO TESTING TOOL

The tool for automated testing of OO programs has the following components/features:

a). Import File Feature

b). Change Impact Identifier for classes

c). Maintenance Tools
d). Logging results
e). Diagram Displayer
f). Class Diagram
g). State Transition Diagram
h). Control Flow Graph

Test Tools:

(i) Test Order generator for testing of classes at integration level
(ii) Test Case generator for testing classes

VII CONCLUSION

This paper deals with Design and Development of an Automated Testing Tool for OO software. The tool mainly focuses on testing design specifications for OO software. An advantage of testing software specifications as compared to program code is that specifications are generally correct whereas code is flawed. Moreover, with software engineering principles firmly established in the industry, nowadays, while developing software all the steps of Software Development Life Cycle (SDLC) are adhered to. For this work, UML specifications are considered. UML has become the defacto standard for analysis and design of OO software. UML designs created in Rational Rose are used by the tool as input. The main components of this tool are:

1. Test Order Generator for classes
2. Test Case Generator for State-based class testing
3. Change Impact Identification for Classes

VIII FUTURE WORK

Future work would include extending the tool to incorporate more functionality. Both testing and maintenance components can be added. Some additions can be:

1. Incorporating a fully functional Method Basis Path Generator module.
2. Providing both Test Case Generation as well as Execution. The user would be able to provide test data; and the test cases generated would be executed using the test data as input.
3. Reporting Code Coverage achieved after Test Set has been executed. Various test adequacy criteria like statement coverage, branch coverage, and path coverage can be reported upon.
4. Metrics: Certain program metrics like Lines of Code(LOC), function points, interfaces, etc. can be reported upon.

REFERENCES

1. Jorgensen, Erikson ”Object-oriented Integration Testing” Communications of the ACM, Vol. 37, No. 9, 1994

2. Kung, Gao, Hsia ”Developing an OO Testing and Maintenance Environment” Communications of the ACM, Vol. 38, No. 10, 1995.

3. Doong, Frankl ”ASTOOT approach to testing OO Programs” ACM Transactions on Software Engineering and Methodology, Vol. 3, No.2, 1994

4. Doong, Frankl ”Case Studies on testing OOprograms” Communications of the ACM, Vol. 25, No. 5, 1991.

5. M. Smith and D. Robson ”A Framework for Testing Object-Oriented programs” Journal of Object-Oriented Programming, June 1994, pp.45 - 53.

6. Frankl, Elaine Weyuker ”An applicable family of data flow testing criteria” IEEE Transactions on Software Engineering, Vol. 14, No. 10, 1988.

7. Mary Jean Harrold, Gregg Rothermel “Performing data flow testing on classes” December 1994 ACM SIGSOFT Software Engineering Notes , Proceedings of the 2nd ACM SIGSOFT symposium on Foundations of software engineering, Volume 19 Issue 5

8. Ugo Buy, Alessandro Orso, Mauro Pezze “Automated Testing of Classes” August 2000 ACM SIGSOFT Software Engineering Notes , Proceedings of the 2000 ACM SIGSOFT international symposium on Software testing and analysis, Volume 25 Issue 5

9. Gao, J.Z.; Kung, D.; Hsia, P.; Toyoshima, Y.; Chen, C. ”Object state testing for object-oriented programs” Computer Software and Applications Conference, 1995. COMPSAC 95. Proceedings., Nineteenth Annual International, 9-11 Aug. 1995 Pages:232 – 238

10. T. Korson and J. D. McGregor. “Understanding object- oriented: a unifying paradigm." CACM Vol. 33, No. 9, pp. 40 - 60, Sept. 1990.

11. D. Kung, J. Gao, P. Hsia, J. Lin and Y.Toyoshima, \Design Recovery for Software Testing of Object-Oriented Programs,Proc. of the Working Conference on Reverse Engineering, pp. 202 - 211, Baltimore Maryland, May 21 - 23, IEEE Computer Society Press, 1993.

12. D. Kung, N. Suchak, P. Hsia, Y. Toyoshima, and C. Chen, \On object state testing," Proc. Of COMPSAC'94, pp. 222 { 227, IEEE Computer Society Press, 1994}.

13. D. Kung, J. Gao, P. Hsia, F. Wen, Y. Toyoshima, and C. Chen, \Change impact identification in object oriented software maintenance," Proc. of IEEE International Conference on Software Main- tenance, pp. 202 - 211, 1994.

14. D. Kung, J. Gao, P. Hsia, Y. Toyoshima, and C. Chen, \A test strategy for object-oriented systems, Proc. of Computer Software and Applications Conference, pp. 239 - 244, Dallas Texas, August 911, IEEE Computer Society, 1995.

15. D. Kung, J. Gao, P. Hsia, Y. Toyoshima, C. Chen, Y.S. Kim, and Y. Song, \Developing an object-oriented software testing and maintenance environment", Communications of the ACM, Vol. 38, No.10, pp. 75 { 87, October 1995}.

16. D. Kung, Y. Lu, N. Venugopalan, P. Hsia, Y. Toyoshima, C. Chen, J. Gao, \Object state testing and fault analysis for reliable software systems," Proc. Of 7th International Symposium on Software Reliability Engineering, White Plains, New York, Oct. 30 - Nov. 2, 1996.
17. D.L. Parnas, \A technique for software module specification with examples," CACM May, 1972.
18. Allen S., Richard B. Borie and David W. Cordes, \Automated Flow Graph-Based Testing of Object-Oriented Software Modules," Journal of Systems Software, no. 23, 1993.
19. D. E. Perry and G. E. Kaiser, \Adequate testing and object-oriented programming," Journal of Object-Oriented Programming, Vol. 2, pp. 13 - 19, January-February 1990.
20. R. Poston, \Automated testing from object models," CACM Vol. 37, No. 9, pp. 48 - 58, September 1994.