OPEN ACCESS

International
Journal
Of Modern Engineering Research (IJMER)

Case Generation from the Combination of UML Class and Activity Diagrams

Er. Seema Saini[1], Er. V.K. Srivastava[2]
(Deptt.of Computer Sci. & Tech.,Haryana College of Technology & Management, Kaithal, India)

ABSTRACT: *Quality software can be developed when it is properly tested. Due to increase in the size and complexity of object-oriented software, manual testing has become time, resource and cost consuming. Properly designed test cases discover more errors and bugs present in the software. The test cases can be generated much early in the software development process, during the design phase. The unified modelling language (UML) is the most widely used language to describe the analysis and designs of object-oriented software. Test cases can be derived from UML models more efficiently. In our work, we propose a novel approach for automatic test case generation from the combination of UML class and activity diagrams. In our approach, we first draw the UML class and activity diagrams using IBM Rational Software Architect (RSA). Then, export the XML metadata interchange (XMI) from IBM Rational Software Architect (RSA). The XMI file is processed to extract variables from the class and predicates from activity diagram using Java code. The predicates are then used to generate the test cases. We have not used any intermediate form which makes the automation difficult. Our approach achieves 100% branch coverage and suitable for mutation testing and unit testing. In our next approach, we focus on UML composite structure diagram to generate test scenarios for integration testing. In our approach, we first draw the UML composite structure diagram using IBM Rational Software Architect (RSA). Then, export the XML metadata interchange (XMI) representation of composite structure diagram from IBM Rational Software Architect. Then, we parse the XMI code and generate the Component Structure Graph (CSG) automatically. Subsequently, we propose two algorithms to generate test scenarios for Top-Down and Bottom-Up integration approach. The generated test scenarios are sufficient enough to find the component in which probability of bug presence is maximum.*

Keywords: *unified modelling language, Test Case, Mutation Testing, Integrated test case generation algorithm.*

I. INTRODUCTION

Software testing usually involves executing a program on a set of tests and comparing the expected output with the actual output [1]. Testing is done to find the errors, which may latter cause system failure. The Testing phase is carried out in three steps: test case generation, test execution and test evolution [2]. Test case generation requires a lot of effort and remaining two steps are relatively easy. Further, due to increase in size and complexity of software, the generation of effective test cases is becoming much more difficult. Manual testing requires a lot of time, cost and most important it is error-prone. So, automated testing is becoming more popular, as it requires less manpower. If the testing process begins before implementation, cost of the software development is reduced. Testing also measures the software quality in terms of its capability for reliability, correctness, maintainability, testability, usability and re-usability. Some of the objectives of testing are as follows:

- A quality test case should have high probability of finding an error.
- It ensures quality of the product.
- Software testing prevents the occurrence of failure.

1.1 Model Based Testing

Model based testing is testing technique in which test cases are derived from a model that describes some (usually functional) aspects of the system under test(SUT). A model is a depiction of a system's behaviour. Models help us understand and envisage the system behaviour. Model based testing involves three steps

1. Creating a model of system requirements for testing.
2. Generating test data from this requirement-model representation.

3. Verifying your design algorithm with generated test cases

A typical deployment of MBT in industry goes through the four stages shown in 1.1. The model is generally created from the requirement specification document.

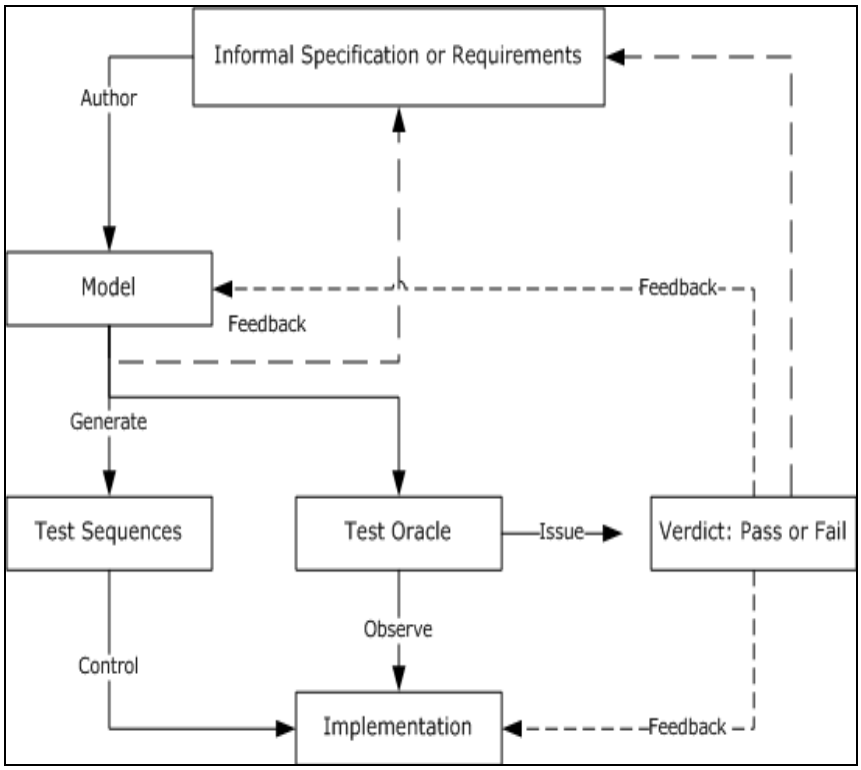

Figure 1.1: Model Based Testing Process

The model is then used to generate the test suites. These test suites contain both the test oracle and test sequence. Test sequence is used to control the system under test. Test Oracle is used for determining whether a test has failed or passed. A failure indicates that the system does not perform according to user requirement.

1.1.1 Benefits of Model Based Testing

1. Model-based testing is easily understood by both the business and developer communities.
2. Model-based testing divides business rationale from testing code.
3. Model-based testing is the quickest approach to get utilization of automated testing.
4. Model-based testing empowers us to switch testing instrument if required or help various stages utilizing the same model.
5. Model-based testing focuses on requirement coverage.
6. Design more and code less.

2.1 Test Case

Test cases are built using specifications and requirements document, i.e., what the system needs to perform. A test case is a triplet (I, S, O) where I is the data input to the system, S is the state of the system to which the data is input, and O is the expected output obtained from the system [4]. Combination of test cases with which a given software product is to be tested is called test suite [5].

2.2 Testing Techniques

Testing techniques are mainly divided into three categories:

2.2.1 Black Box Testing Technique

Black-box testing examines the functionality of an system without having the knowledge of internal logic of code. In a black box testing the tester only knows the inputs and what the expected outcomes should be and not how the program arrives at those outputs. It is also known as functional testing.

2.2.2 White Box Testing Technique

In white box testing test cases are designed based on analysis of some aspect of source code and is based on some heuristic. White box testing is also called glass testing, open box testing and structural testing. To perform white box testing on a application, the analyser needs to have learning of the inward working of the code.

2.2.3 Grey Box Testing Technique

Grey black box testing is a combo of black box testing and white box testing. The analyser has the constrained learning of the inside workings of a application. It is focused around the interior information structures and calculations for planning the experiments more than black box testing however short of what white box testing. This system is imperative when directing integration testing between two modules of code composed by two separate developers, where just interfaces are uncovered for test.

II. RELATED WORK

Wang et al. [4] proposed an approach to generate the test case from an interaction and class diagrams. They used test adequacy criteria for the coverage of the design model elements. They have adopted the category partition approach to get the function units, then for each function unit, generate test cases from class diagram criteria. method is introduced by Asthana et al. [5] for generating test cases using class and sequence diagrams. First, they get the lower and upper bound of variable from the given class diagram. Then, they have traversed the sequence diagram to obtain the all variable passed. Out of these variables, they found out the variables on which the output will derive and have applied robustness testing on these variables to compare the results.

A method is proposed by Swain et al. [6] to generate test case based on use case and sequence diagram. They constructed Concurrent Control Flow Graph (CCFG) from sequence diagrams and Use case Dependency Graph (UDG) from use case diagram to generate test sequence. They have used UML 2.0 sequence diagram for generating test cases. They have developed a semi automated tool (ComTest) which takes XMI representation of sequence diagram as input and generate the test cases. Their testing strategies to derive test cases use full predicate coverage criteria.

A methodology is proposed by Swain et al. [7] to prioritize test scenario from UML communication and activity diagrams. They presented an integrated approach and a prioritization technique to generate cluster-level test scenarios from UML communication and activity diagrams. First, they convert the communication and activity diagrams into a tree representation respectively. Then combine the tree representation of diagrams into intermediate tree named as COMMACT tree. The COMMACT tree is then traversed to generate the test scenarios. They have proposed a prioritization metric considering the coupling or impact or influence of activity and methods.

Pilskalns et al. [8] presented a graph based approach to combine the information form sequence diagrams and class diagrams. In this approach, first sequence diagram is transformed into an object-method directed acyclic graph (OMDAG). The values of variable in class diagram are then associated with objects in OMDAG during path traversal. The execution sequence and attribute value of generated test cases is stored into an object method execution table (OMET).

III. CONCLUSION

Model based testing is testing technique in which test cases are derived from a model that describes some (usually functional) aspects of the system under test(SUT). A model is a depiction of a system's behaviour. Model-based testing is easily understood by both the business and developer communities. It divides business rationale from testing code. It is the quickest approach to get utilization of automated testing. Model-based testing empowers us to switch testing instrument if required or help various stages utilizing the same model. This technique of testing focuses on requirement coverage.

REFERENCES

[1]. P. Ammann and J. Offutt, Introduction to software testing Cambridge University Press, 2008.
[2]. P. N. Boghdady, N. L. Badr, M. Hashem, and M. F. Tolba, "A proposed test case generation technique based on activity diagrams.," International Journal of Engineering & Technology, vol. 11, no. 3, 2011.
[3]. J. Arlow and I. Neustadt, UML 2 and the unified process: practical object-oriented analysis and design. Pearson Education, 2005.
[4]. Y. Wang and M. Zheng, "Test case generation from uml models," in 45th Annual Midwest Instruction and Computing Symposium, Cedar Falls, Iowa, vol. 4, 2012.
[5]. S. Asthana, S. Tripathi, and S. K. Singh, "A novel approach to generate test cases using class and sequence diagrams," in Contemporary Computing, pp. 155{167, Springer, 2010.
[6]. S. K. Swain, D. P. Mohapatra, and R. Mall, "Test case generation based on use case and sequence diagram," International Journal of Software Engineering, IJSE, vol. 3, no. 2, pp. 21{52, 2010.

[7]. R. K. Swain, V. Panthi, D. P. Mohapatra, and P. K. Behera, "Prioritizing test scenarios from uml communication and activity diagrams," Innovations in Systems and Software Engineering, pp. 1{16, 2013.
[8]. O. Pilskalns, A. Andrews, S. Ghosh, and R. France, "Rigorous testing by merging structural and behavioral uml representations," in UML The Unified Modeling Language. Modeling Languages and Applications, pp. 234{248, Springer, 2003.
[9]. S. Kansomkeat, P. Thiket, and J. Offutt, "Generating test cases from uml activity diagrams using the condition-classification tree method," in Software Technology and Engineering (ICSTE), 2010 2nd International Conference on, vol. 1, pp. V1{62, IEEE, 2010.
[10]. S. K. Swain, S. K. Pani, and D. P. Mohapatra, "Model based object-oriented software Testing.," Journal of Theoretical & Applied Information Technology, vol. 14, 2010.
[11]. M. Aggarwal and S. Sabharwal, "Test case generation from uml state machine diagram: A survey," in Computer and Communication Technology (ICCCT), 2012 Third International Conference on, pp. 133{140, IEEE, 2012.
[12]. Y. Le Traon, T. Jeron, J.-M. Jezequel, and P. Morel, "Efficient object-oriented integration and regression testing," Reliability, IEEE Transactions on, vol. 49, no. 1, pp. 12{25, 2000.
[13]. V. Le Hanh, K. Akif, Y. Le Traon, and J.-M. Jezeque, "Selecting an efficient oo integration testing strategy: an experimental comparison of actual strategies," in ECOOP Object-Oriented Programming, pp. 381{401, Springer, 2001.
[14]. P. C. Jorgensen and C. Erickson, "Object-oriented integration testing," Communications of the ACM, vol. 37, no. 9, pp. 30{38, 1994.
[15]. W. E. Howden, "Weak mutation testing and completeness of test sets," Software Engineering, IEEE Transactions on, no. 4, pp. 371{379, 1982.
[16]. M. Sarma, D. Kundu, and R. Mall, "Automatic test case generation from uml sequence diagram," in Advanced Computing and Communications, 2007.ADCOM 2007. International Conference on, pp. 60{67, IEEE, 2007.
[17]. Y. Wu, M.-H. Chen, and J. Offutt, "UML-based integration testing for component-based software," in COTS-Based Software Systems, pp. 251{260, Springer, 2003.
[18]. http://www.rspa.com/spi/test-methods.html
[19]. http://www.slideshare.net/angoes12/software-engineering-a-practitioners-approach-roger-s-pressman
[20]. http://www.rspa.com/
[21]. Jim Arlow, Ila Neustadt, "UML 2 and the Unified Process – Practical Object Oriented Analysis and Design", Pearson Education.
[22]. Bernd Bruegge, Allen H. Dutoit, "Object Oriented Software Engineering using UML", Pearson Education.