Study Of Approaches For Generating Automated Test Cases By UML Diagrams

N. K. Sharma
Assistant Professor
Rajiv Gandhi Technical University

Divya Saxena
M.Tech Student
Jayoti Vidyapeeth Women's University

Abstract

Software testing is a crucial phase in the Software Development Life Cycle (SDLC). Testing includes executing a program on a set of test cases and comparing the actual results with the ones expected. Test cases describe tests that need to be run on the program to verify that the program runs as expected. To reduce the time consumption and to cut down the cost of manual testing researchers and practitioners have proposed various tools and techniques for automation of software testing which increases the reliability of the software. In general, the software testing phase takes around 40-70% of the time and cost during the software development life cycle. To test the software automatically, test case generation is the best way. One way to generate the test cases is with the help of UML diagrams. In this paper we study the various approaches used to generate the test cases from the UML diagrams to test the software automatically.

Keywords: Test case, UML, Automatic test case generation, Model based testing.

1. Introduction

Software testing is an important activity in software development life cycle. It is an investigation activity conducted to provide all stakeholders with information about the quality of given software or applications. Software organizations spend considerable portion of their budget in testing related activities. A well tested software system will be validated by the customer before acceptance. While developing the software, the software organizations spend near about 50% of their budget in testing related task. It provides the efficiency of the software and the correctness of the software. Testing can be done either manually or automatically. Testing automatically is best way to test software because it consume less time and give accurate result than manual testing. A test case in software engineering is a set of conditions or variables under which a tester will determine whether an application or software system is working correctly or not. Test case is generated from the UML using object diagrams. UML is unified modeling language that is used to create visual models of a software system. These models can help to create designs and to permit analysis and review of these models. There are different types of diagram in the UML depicting the dynamic behavior of objects in a system. Unified Modeling Language is a widely accepted set of notations for modeling object oriented system. Different techniques are used which uses different approaches to generate the test cases from object diagram.

2. Literature Survey

This section discusses the work done related to generating the test cases automatically to test the software. There are various techniques with different method to generate the test cases. Several researches have successfully proposed test case generation for various software under various circumstances such as scenario-based, model based, path oriented, goal-oriented and genetic approaches. This section surveys and describes the waterfall **software development life cycle**, **software testing process**, **test case generation process** and all recent research of test case generation techniques.
Since Unified Modeling Language (UML) is a standardized general-purpose modeling language in the field of software engineering. UML diagrams represent two different views of a system model: static and dynamic. Static (or *structural*) view: emphasizes the static structure of the system using objects, attributes, operations and relationships. The structural view includes class diagrams and composite structure diagrams. Dynamic (or *behavioral*) view: emphasizes the dynamic behavior of the system by showing collaborations among objects and changes to the internal states of objects. This view includes sequence

diagrams, activity diagrams and state machine diagrams. According to the waterfall software development life cycle (SDLC) below, basically there are five phases in the cycle, which are:
(a) Requirements
(b) Design
(c) Implementation (also known as development)
(d) Verification (also known as software testing)
(e) Maintenance.

Software testing phase is the process of executing a program or system with the intent of finding errors. [1]. It involves any activity aimed at evaluating an attribute or capability of a program or system and determining that it meets its required results [2]. Software testing is an essential activity in the SDLC. In the simplest terms, it provides quality assurance by observing the execution of a software system to validate whether it behaves as intended and to identify potential malfunctions. Earlier studies estimated that testing can consume fifty percent, or even more, of the development costs [3], while a detailed survey in the United States [4] quantified the high economic impacts of an inadequate software testing infrastructure. The software testing process provided by Pan [5] from Carnegie Mellon University, describe the general process of running software testing activities.

2.1. Requirements analysis: Software testing should begin in the requirements phase of the SDLC. During the design phase, software testing engineers work with developers in determining what aspects of a design are testable and with what parameters those tests work.

2.2. Test planning: Test strategy, test plan, testbed creation. A testbed is a platform for experimentation for large development projects. Testbeds allow for rigorous, transparent and replicable testing of scientific theories, computational tools, and other new technologies.

2.3. Test development: Develop test procedures, design test scenarios, produce test cases, prepare test datasets, and build test scripts to use in testing software

2.4. Test execution: Once the test plan and test cases, including test data, are generated and prepared, software testing engineers can execute the software based on the plans and tests and report any errors found to the development team.

2.5. Test reporting: When the test cases have been run, software testing engineers generate metrics and make final reports on their test effort and whether or not the software tested is ready for release.

2.6. Test result analysis (also known as defect analysis): This step is done by the testing team. It is usually done along with the client, in order to decide what defects should be treated, fixed, rejected (i.e. found software working properly) or deferred to be dealt with at a later time.

2.7. Retesting the resolved defects: When a defect has been resolved by the development team, the test must be run again.

2.8. Regression testing: In general, it is common to have a small test program built based on a subset of tests, for each integration of new, modified or fixed software, in order to ensure that the latest delivery has not ruined anything. Additionally, this step ensures that the software product as a whole is still working correctly.

2.9. Test Closure: When the test meets the exit criteria, the activities such as capturing the key outputs, lessons learned, results, logs, documents related to the project are archived and used as a reference for future projects.

Model based technique identify respective test case for the software with respect to the UML diagrams such as activity, state-chart, object diagram etc. Path-oriented testing based on static as well as dynamic control flow of the software. Static path testing is done by symbolic execution, dynamic path testing based on the run time test of executing program. Goal-oriented techniques identify test cases covering a selected goal such as a statement or branch, irrespective of the path taken.
M.Prasanna *et al.* (2009) presents that to test the software, test cases generation is best way. The test cases are derived by analyzing the dynamic behavior of the objects due to internal and external stimuli [6]. Researchers use model based approach in which genetic algorithm’s crossover technique is apply on the class diagram and the traversal is done by the depth first search(DFS) algorithm. This tree structure approach coupled with genetic algorithm shows that it is capable to reveal 80% faults in unit level and 88% faults in integration level. They couple the genetic algorithm with mutation testing to check the effectiveness in the testing process which shows 80.3% of effectiveness.

A.V.K Shanth *et al.* (2011) have proposed another model based approach in which the concept of data mining is used in which the evolutionary genetic algorithm technique is apply on the class diagram and generate the test cases [7]. They show that evolutionary genetic algorithm yields optimal valid test case than with only genetic crossover operator, after applying depth first searching algorithm. The advantages are that specification –based testing uses information derived from a specification to assist testing as well as to develop program.

G.Mohan Kumar, A.V.K.Shanthi (2012) researchers used some novel approaches to test the software at the initial stage itself which will make easy for the software testers to test the software in the later stage [8]. Here they take the sequence diagram. The experiment results show that this method has better performance. All the possible test cases are generated and validated by prioritization.

Sangeeta sabhwal, Ritu sibal, Chayanika Sharma (2011) consider in their paper another novel based approach in which testing efficiency is optimized by applying the genetic algorithm on the test data. For requirement change, a stack based approach for assigning weights to the nodes object diagram is proposed [9]. Here first sequence diagram is generated and then from the sequence diagram, sequence dependency graph is generated and genetic algorithm is applied on it [10].

Ranjit Swain, Vikas Panthi, Durga Prasad (2012) used functional minimization technique to generate the test cases. In this technique in which the STUPEC [11] technique is used in which first predicate is selected and then predicated is transformed and then test cases are generated. The functional minimized technique is used for finding the minimum of predicate function. In this approach the test cases are generated step by step. Here the object diagram that is used for generating the test cases is state machine diagram. This approach covers much coverage like state coverage, transition pair coverage, action coverage. The numbers of test cases are minimized that achieve transition path coverage by testing the borders determine by simple prediction.

L.C.Briand *et al.* (2008) in his paper proposed the method supported by the prototype tool to tackle the regression test selection problem at the design level. The main objective has been to ensure that regression testing was safe while minimizing regression efforts.

Alessandra Cavarra, Thierry Jeron, Alan Hartman *ISSTA* (2002) present an architecture for model-based testing using a profile of the unified modeling language (UML). Class, object and state diagrams are used to define essential model. To generate the test cases automatically, generation tool like AGEDIS test is used [12]. The AGEDIS test generation tool is based on the principles of two exiting tools that are TVG and GOTCHA. The main advantage of the AGEDIS test case generation tool is its ability to combine different test directives: coverage criteria, test purposes and test constraints.

3. Test Case Generation Techniques

Test case generation has always been an integral part to the testing process. There are many types of test case generation techniques [13] such as specification-based techniques, sketch diagram based techniques and source code-based techniques. Random techniques determine a set of test cases based on assumptions concerning fault distribution. Source code-based techniques generally use a control flow graph to identify paths to be covered and generate appropriate test cases for those paths. Goal-oriented techniques identify test cases covering a selected goal such as a statement or branch, irrespective of the path taken.

3.1. Specification-Based Techniques

Specification-based techniques are methods to generate a set of test cases from specification documents such as a formal requirements specification [14], [15], [16], [17]. The specification precisely describes what the system is to do without describing how to do it. The advantages of this technique include that the specification document can be used to derive expected results for test data, and that tests may be developed concurrently with design and implementation. The specification requirement document can be used as a basis for output checking, significantly reducing one of the major costs of testing. Specifications can also be analyzed with respect to their testability [18].Furthermore, the specification-based technique offers a simpler, structured, and more formal approach to the development of functional tests than non-specification based testing techniques do. The strong relationship between specification and tests helps find faults and can simplify regression testing. The process of generating tests from the specifications will often help the test engineer discover problems with the specifications themselves. If this step is done early, the problems can be eliminated early, saving time and resources.

3.2. Sketch diagram-Based Techniques

Sketch diagram-based techniques are methods to generate test cases from model diagrams like UML Use Case diagram [19], [20], [21], [22] and UML State diagrams [23], [24], [25], [26]. A major advantage of model-based V&V is that it can be easily automated, saving time and resources. Other advantages are shifting the testing activities to an earlier part of the software development process and generating test cases that are independent of any particular implementation of the design [19]. The sketch diagram-based test case generation techniques have been proposed for traditional and web-based application for a long time. Jim [20] presented how using use cases to generate test cases can help launch the testing process early in the development lifecycle and also help with testing methodology. In a software development project, use cases define system software requirements. Use case development begins early on, so real use cases for key product functionality are available in early iterations. Web based applications are of growing complexity and it is a serious business to test them correctly. Manish [22] focused on black box testing which enables the software testing engineers to derive sets of input conditions that will fully exercise all functional requirements. They believed that black box testing is more generally suitable and more necessary for web applications than other types of application.

3.3. Source Code-Based Techniques

Source code-based techniques generally use control flow information to identify a set of paths to be covered and generate appropriate test cases for these paths. The control flow graph can be derived from source code. The result is a set of test cases with the following format:
a) test case ID
b) test data
c) test sequence (also known as test steps)
d) expected result
e) actual result and
f) pass / fail status.
Source code based techniques generally use control flow information to identify a set of paths to be covered and generate appropriate test cases for these paths. These techniques can further be classified as static or dynamic. Static techniques are often based on symbolic execution e.g. [27], whereas dynamic techniques obtain the necessary data by executing the program under test e.g. [28].

4. Test Case Generation Approaches

Even though variety of approaches have been proposed, with the advent of modeling tools like Rational Rose, for a decade there has been constant research on generating test cases based on specifications and design models. Approaches involved in generating of test case can be categorized in scenario-based, model-based and genetic-based test case generation.

4.1. Scenario-Based Test Case Generation

Scenario-based test cases generation is based on concurrent application availed in used case studies. Baikutha Narayan Biswal [23] had presented paper on “A Novel Approach for Scenario-Based Test Case Generation”. They have proposed scenario-based testing, test scenarios are used for generating test cases. UML activity diagrams describe the realization of the operation in design phase and also support description of parallel activities and synchronization aspects involved in different activities perfectly. This paper deals with test adequacy criteria. Scenario testing works best for complex transactions or events, for studying end-to-end delivery of the benefits of the program, for exploring how the program will work in the hands of an experienced user, and for developing more persuasive variations of bugs found using other approaches.
Concurrent Application testing described by Chang-ai Sun [29] also employs UML Activity diagram for generating the test case. A paper on A Transformation-based Approach to Generating Scenario-oriented Test Cases from UML Activity Diagrams for Concurrent Applications promotes transformation-based approach to generating scenario-oriented test cases for testing concurrent applications modeled by UML Activity Diagrams. Concurrent behavior is nondeterministic
its testing is more difficult than the testing of common control flows or data flows.
Test Cases Generation from UML Activity Diagrams presented by Hyungchoul Kim[30] also based on concurrency in Activity Diagram i.e., concurrent system in which multiple objects interact with each other. The proposed method generates test cases from UML activity diagrams that minimize the number of test cases generated while deriving all practically useful test cases.
Test Case Design Using Conditioned Slicing of Activity Diagram by Mitrabinda Ray, Soubhagya Sankar Barpanda, and Durga Prasad Mohapatra presents conditioned slicing as a general slicing framework for test case generation from activity diagram [31]. The Method first builds a flow

dependence graph from an ordinary UML activity diagram and then applies conditioned slicing on a predicate node of the graph, to generate test cases.

4.2. Model-Based Test Case Generation

Model based test case generation equally challenging and also many researches involve achieving optimal set of test case. Automatic test case generation using unified modeling language (UML) state diagrams by P. Samuel, R. Mall, and A.K. Bothra published on basis of model based test case generation automatically. The approach, the control and data flow logic available in the UML state diagram to generate test data are exploited. The state machine graph is traversed and the conditional predicates on every transition are selected. Then these conditional predicates are transformed and function minimization technique is applied to generate test cases. The present test data generation scheme is fully automatic and the generated test cases satisfy transition path coverage criteria. The generated test cases can be used to test class as well as cluster- level state-dependent behaviors [32].

Test Case Generation by means of UML Sequence Diagrams and Labeled Transition Systems by Emanuela G.Cartaxo, introduced UML Sequence diagram in field of test case generation. A feature is an increment of functionality, usually with a coherent purpose that is added on top of a basic system. Feature are usually developed and tested separately from the basic system as independent modules. The procedure is based on model-based testing techniques with test cases generated from UML sequence diagrams translated into Labeled Transition Systems (LTSs) [33]. The idea is to reuse sequence diagrams that are constructed by development teams to specify use cases with basic and alternative scenarios. Model-based software development bases on setting up models of the system to be constructed. This approach has proved to be useful, because it allows developers to first elaborate the most important properties of the software before proceeding with the implementation.

In Test Case Generation from UML State Machines presented by Dirk Seifert elaborates test cases include not only stimuli to trigger the system under test, they also include possible correct observations to automatically evaluate the test case ecution.[34].

Automated-Generating Test Case Using UML Statecharts Diagrams by Supaporn Kansomkeat and Wanchai Rivepiboon experimented on the automatic testing technique to solve partially the testing process. This technique can automatically generate and select test cases from UML state chart diagrams. Firstly, transform this diagram into intermediate diagram, called Testing Flow Graph (TFG), explicitly identify flows of UML state chart diagrams and enhance for testing. Secondly, from TFG generate test case using the testing criteria that is the coverage of the state and transition of diagrams. Finally, the evaluation is performed using mutation analysis to assess the fault revealing power of our test cases [35].

Test Case Generation Based on Use case and Sequence Diagram by Santosh Kumar Swain, Durga Prasad Mohapatra, and Rajib Mall illustrate Test cases are derived from analysis artifacts such as use cases, their corresponding sequence diagrams and constraints specified across all these artifacts. Construct Use case Dependency Graph (UDG) from use case diagram and Concurrent Control Flow Graph (CCFG) from corresponding sequence diagrams for test sequence generation. Focus testing on sequences of messages among objects of use case scenarios [36].

4.3. Genetic-Based Test Case Generation

Automatic Test Case Generation for UML Object diagrams using Genetic Algorithm presented by M. Prasanna and K.R. Chandran used to generate optimal test cases which also can be consider as data mining approach.

Automated generation of test cases in object oriented systems has been presented. The test cases are derived by analyzing the dynamic behavior of the objects due to internal and external stimuli .The scope of the paper has been limited to the object diagrams taken from the Unified Modeling Language model of the system. Genetic Algorithm’s tree crossover has been proposed to bring out all possible test cases of a given object diagram [12]. Experimental results show that it has the capability to reveal 80% fault in the Unit level and 88% fault in the integration level.

5. Conclusion

To sum up all, there are various techniques available for generating test cases to satisfy test coverage as well as path coverage criteria. Scenario-Based test case generation mainly focused on concurrent process in only activity diagram. Model-Based test case generation focused on various state charts, sequence, object, use case diagrams to generate test case but fails produce optimal one. Genetic based test case generation produce an optimal on but still fault test cases are available. In recent trend Model-Based test case attracts many researchers by using some data mining concept to produce an automated optimal test case by which minimum human and cost effort are utilized. There is still a great scope for future work to find improvements in current testing techniques.

6. References

[1] Myers, Glenford J., "The art of software testing", Publication info: New York : Wiley. ISBN: 0471043281, 1979.
[2] Hetzel, William C., "The Complete Guide to Software Testing", 2nd ed. Publication info: Wellesley, Mass.: QED Information Sciences. ISBN: 0894352423, 1988.
[3] B. Beizer, "Software Testing Techniques", Van Nostrand Reinhold, Inc, New York NY, 2nd edition. ISBN: 0-442-20672-0, 1990.
[4] NIST, "The economic impacts of inadequate infrastructure for software testing", 2002.
[5] Pan, Jiantao, "Software Testing (18-849b Dependable Embedded Systems)", Electrical and Computer Engineering Department, Carnegie Mellon University, 1999.
[6] L.C.Briand, Y. Labiche, S.He, "*Automating Regression Test Selection based on UML designs*", Information and Software Technology 51(2009)16-30.
[7] Sangeeta Sabharwal, Ritu Sibal, Chayanika Sharma, " *Applying Genetic Algorithm for Prioritization of Test Case Scenarios Derived from UML Diagrams*", International Journal of Computer Science Issues, vol.8, issue 3, no.2, May 2011, pp. 433-444.
[8] A.V.K.Shanthi, G.Mohan Kumar, " *Automated Test Cases Generation from UML Sequence Diagram*", International Conference on Software and Computer Application, vol.41, 2012, pp. 83-89.
[9] Marlon Vieira, Johanne Leduc, Bill Hasling, Rajesh Subramanyan , Juergen Kazmeier, " *Automation of GUI Testing Using a Model – Driver Approach*", Proceeding of International Workshop on Automation of Software Test, 2006, pp. 9-14.
[10] Philp Samuel, R.Mall, A.K.Bothra, "*Automatic Test Case Generation Using UML State Diagram*", IET Software, 2008, pp. 79-93.
[11] A.V.K.Shathi, G.Mohan Kumar, " *A Heuristic Approach for Automated Test Case Generation From Sequence Diagram Using Tabu Search Algorithm*", European Journal of Scientific Research, vol.85, no.4, Sep 2012,pp. 534-540.
[12] M. Prasanna, K.R.Chandran, "*Automatic Test Case Generation for UML Object Diagrams Using Genetic Algorithm*", Int. J. Advance. Soft comput. Appl., vol.1, no. 1, July 2009, pp. 19-32.
[13] M. Prasanna S.N. Sivanandam R.Venkatesan R.Sundarrajan, "A Survey on Automatic Test Case Generation", Academic Open Internet Journal, 2005.
[14] Hung Tran, "Test Generation using Model Checking", Proceeding Conference on Automated Verification, 2001.
[15] Miao Huaikou and Liu Ling, "A Test Class Framework for Generating Test Cases from Z Specifications", 2000.
[16] Percy Antonio, Pari Salas and Bernhard K. Aichernig, "Automatic Test Case Generation for OCL: a Mutation Approach", 2005.
[17] Richard A. DeMillo and A. Jefferson Offutt, "Constraint-Based Automatic Test Data Generation", IEEE Transaction on Software Engineering, 1991.
[18] Aynur Abdurazik and Jeff Offutt, "Generating Test Cases from UML Specifications", 1999.

[19] A.Z. Javed, P.A. Strooper and G.N. Watson, "Automated Generation of Test Cases Using Model-Driven Architecture", Second International Workshop on Automation of Software Test (AST'07), 2007.
[20] Jim Heumann, "Generating Test Cases From Use Cases", Rational Software, 2001.
[21] Johannes Ryser and Martin Glinz, "SCENT: A Method Employing Scenarios to Systematically Derive Test Cases for System Test", 2000.
[22] Manish Nilawar and Dr. Sergiu Dascalu, "A UML-Based Approach for Testing Web Applications", Master of Science with major in Computer Science, University of Nevada, Reno, 2003.
[23] Alessandra Cavarra, Charles Crichton, Jim Davies, Alan Hartman, Thierry Jeron and Laurent Mounier, "Using UML for Automatic Test Generation", Oxford University Computing Laboratory, Tools and Algorithms for the Construction and Analysis of Systems, TACAS'2000, 2000.
[24] Annelises A. Andrews, Jeff Offutt and Roger T. Alexander, "Testing Web Applications. Software and Systems Modeling", 2004.
[25] Avik Sinha, Ph.D and Dr. Carol S. Smidts, "Domain Specific Test Case Generation Using Higher Ordered Typed Languages from Specification", Ph. D. Dissertation, 2005.
[26] David C. Kung, Chien-Hung Liu and Pei Hsia, "An Object-Oriented Web Test Model for Testing Web Applications", In Proceedings of the First Asia Pacific Conference on Quality Software (APAQS'00), page 111, Los Alamitos, CA, 2000.
[27] C. Ramamoorthy, S. Ho, and W. Chen, "On the automated generation of program test data", IEEE Transactions on Software Engineering, SE-2(4):293–300, 1976.
[28] Bogdan Korel, "Automated Software Test Data Generation", IEEE Transaction on Software Engineering, 1990.
[29] Chang-ai Sun, 2008 IEEE, "Transformation-based Approach to Generating Scenario-oriented Test Cases from UML Activity Diagrams for Concurrent Applications", Annual IEEE International Computer Software and Applications Conference.
[30] Hyungchoul Kim, Sungwon Kang, Jongmoon Baik, Inyoung Ko, "Test Cases Generation from UML Activity Diagrams ", Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing.
[31] Mitrabinda Ray, Soubhagya Sankar Barpanda, Durga Prasad Mohapatra, "Test Case Design Using Conditioned Slicing of Activity Diagram", International Journal of Recent Trends in Engineering, Vol. 1, No. 2, May 2009.
[32] P. Samuel, R. Mall, A.K. Bothra, "Automatic test case generation using unified modeling language (UML) state diagrams ", Published in IET Software.

[33] Emanuela G. Cartaxo, Francisco G. O. Neto and Patr´ıcia D. L. Machado, "Test Case Generation by means of UML Sequence Diagrams and Labeled Transition Systems", IEEE 2007.
[34] Dirk Seifert, "Test Case Generation from UML State Machines", inria-00268864, version 2 - 23 Apr 2008 .
[35] Supaporn Kansomkeat and Sanchai Rivepiboon, "Automated- Generating Test Case Using UML Statechart Diagrams ",SAICSIT 2003.
[36] Santosh Kumar Swain, Durga Prasad Mohapatra, and Rajib Mall, "Test Case Generation Based on Use case and Sequence Diagram", Int.J. of Software Engineering, IJSE Vol.3 No.2 July 2010.