International Journal of Computer Applications (0975 – 8887)
Volume 95– No.20, June 2014

UML based Test Case Generation Methods: A Review

Neha Pahwa
M.Tech Student
University Institute of Engg and Technology
M.D University, Rohtak, India

Kamna Solanki
Assistant Professor
University Institute of Engg and Technology
M.D University, Rohtak, India

ABSTRACT
Testing guarantees the quality of software to be developed in terms of presence of errors. A difficult part of software testing entails the generation of test cases. A good test case should have the quality to cover more features of test objective. There are number of methods for test case generation. The use of a model to describe the behavior of a system is a proven and major advantage to test. In this paper, various test case generation techniques based on UML (Unified Modeling Language) are explained. The focus will be on effective use of UML techniques and test-case generation in order to make suitable executions.

General Terms
Software Testing: Software Testing is any process or activity aimed at evaluating a system or attribute or capability of a program and determining through the purpose to find that whether it satisfies or meets the specified requirements or not.

Keywords
Test case; test case generation; UML Diagrams

1. INTRODUCTION
The evolution of computer-based systems and products in the current scenario of globalization is derived from software which is one of the most important technologies and has grown from being a mere problem solving tool, a lot is yet to be done on the development of quality software that performs the right job at the right time. It is here that software engineering intends to provide a structured framework for building high quality software **[**1**]**.It is with this intention that the Software Development Life Cycle (SDLC) is framed, which is a series of steps that are to be followed in an order to produce efficient software which is cheaper . Amongst the different steps in SDLC, software testing is a very important yet a mandatory step, which ensures the proper working of the software.

For evaluating a system or attribute or capability of a program software testing is the capable process and also for determining through the purpose to find that whether it satisfies or meets the specified requirements or not. In simple words testing is executing a system in order to discover any errors gaps or missing requirements in contrary to the actual requirements desire output[**2**]. Software testing consists of following steps: design of test cases, preparing the test data, execution of program with test data, and the last but not the least comparing the result with that of test case. Design or generation of test cases is a challenging part.

To pen down entire coverage to the application and test all possible combinations in the application the user can be helped by test cases. A test case is a pair consisting of test data to be input to the program and the expected output. Whereas a test suite is a collection of test cases that are grouped for test execution purposes. Test case generation plays a vital role in the engineering of test harnesses. In particular, specification-based or black-box test case generation is a mainstream approach, in which test cases are generated according to information derived from the specification without the need to know the implemented code. Another approach for test case generation is white box testing in which testing of implemented code is done. Generating test cases early helps test engineer to find ambiguities and inconsistencies in the requirements specification and design documents [**3**]. The test case generation or the process of designing test is the first and the most important process in software testing [**4**].

The various types of test case generation techniques are random approaches, goal- oriented technique, specification-based techniques, sketch diagram based techniques and source based techniques[**5**]. UML diagrams derive test cases in Sketch diagram based techniques. The UML diagrammatic technique is commonly used in the software design phase. Many diagrams are used in generating a set of test cases, such as use case diagram, activity diagram, class diagram and state chart diagram[**6**]. Model-based testing which uses UML (Unified Modeling Language) design specifications for test case generation overcomes the shortcomings that the system state information is very difficult to identify, either from the requirement specifications or from the code, and has emerged as a promising testing method.

2. RELATED WORK
H.S Hong et al. presented a specification-based approach to class testing using UML state diagrams. They proposed a transformation method from UML state diagrams into flow graphs and showed that conventional flow analysis techniques can be applied to test case generation from UML state diagrams. Using this transformation, they flatten the hierarchical and concurrent structure of states and broadcast communication while preserving both control and data flow in UML state diagrams. The resulting set of test cases provides the capability of checking that classes are correctly implemented against specifications written in UML state diagrams testing whether class implementations established the desired control and data flow specified in the specification [**7**].

Aditya P. Mathur et al. provides a new program execution based approach to generate test data for a given branch in a program is presented in this paper. The method dynamically switches to a path that offers relatively less resistance to generation of an input to force execution through it to reach the given branch. the method is general and applicable to programs with general non linear expressions. Being execution based, it can handle different programming language features. It is suitable for automation and has potential to provide a practical solution to the test data generation problem [**8**].

A. Bertomated et al. proposed a method to generate test cases using the UML use case and interaction diagrams. It basically aims at integration testing to verify that the pretested system components interact correctly. They use category partition method and generate test cases manually following the sequences of messages between components over the sequence diagram [**9**].

M. Sarma et al. presented an approach of generating test cases from UML design diagrams. A UML use case diagrams transformed into a graph called use case diagram graph (UDG) and sequence diagram into a graph called the sequence diagram graph (SDG) and then integrating UDG and SDG to form the System Testing Graph (STG). The STG is then traversed to generate test cases for system testing. They have used state-based transition path coverage criteria for test case generation. Having stored all essential information for test generation in the STG, they now traverse the STG to generate test cases. The Test Suite Generation algorithm, traverse the STG at 2 levels. The traversal begins with the UDG. This traversal visits all use cases and generate test cases for detecting initialization faults. At level 1, if a use case initialization faults occur then it was assume faults in its operation and therefore no need to apply test cases corresponding to the operation. At level 2 traversal, starting from a use case node the corresponding SDG was visited and test cases were generated to detect operational faults [**10**].

Chen et al. proposed a technique in which they use UML activity diagrams as design specifications, and consider the automatic approach to test case generation. Instead of deriving test cases from the UML activity diagram directly, they presented an indirect approach which selects the test cases from the set of the randomly generated test case according to a given activity diagram. In this method, they first randomly generate abundant random test cases. Then, by running the program with these test cases, they will get the corresponding program execution traces. Last, by comparing these traces with the activity diagram according to the specific coverage criteria, they can prune some redundant test cases and get a reduced test case set which meets the test adequacy criteria[**11**].

D. Samanta et al, proposed an approach for generating test cases using UML 2:0 activity diagrams. In this, they considered a coverage criterion called activity path coverage criterion. Generated test suite following activity path coverage criterion aims to cover more faults like synchronization faults, faults in a loop than the existing work [**12**].

R. Mall et al. proposed a novel technique, which generates the test cases from the UML use case diagrams and sequence diagrams in which test requirements and coverage criteria are derived from UML models[**13**]. It covers all the specification requirements.

Chandran et al. proposed a model to generate test cases for the object diagrams. In this methodology various steps were evolved. Object diagrams were constructed, stored and object are named. The tree was build using these object names which further applied with crossover operator of GA. These new generation trees are converted into binary tree and traversed using DFS technique and this gave all the valid, invalid and termination sequences for a given application [**14].**

A comparative evaluation of test cases generated from different UML diagrams is done to understand the roles of different UML diagrams in test case generation. To achieve this goal, test cases that are generated from UML state-charts and sequence diagrams are used both at unit and integration level testing, and their fault revealing capabilities are compared. This experiment is designed from the perspective of a researcher, and is carried out as a case study (single project) by [**15].**

R. Mall et al., presented a technique that enhances the integration testing of classes by accounting for all possible states of interacting objects. They proposed a new intermediate representation named state-activity-diagram (SAD). In SAD, the control flow information during the execution of a use case is shown through a combination of state transitions and activities. It is derived by synthesizing UML state-chart diagrams of different objects involved in a particular use case with an activity diagram. The states are extracted from the state-chart diagrams and control flow is extracted from the activity diagram. To handle concurrent execution, some new types of nodes have been introduced [**16**].

P. Samuel et al., proposed a method to generate test cases by applying dynamic slicing on UML sequence diagrams. In this, they describe how dynamic slicing can be used for test case generation in the object oriented context [**17**].

Y. Wang et al., proposed a methodology to generate the test case from a design level class diagram and an interaction diagram. A car rental example is used to illustrate the test case generation [**18**] .

H.S Hong et al., presented a specification-based approach to class testing using UML state diagrams. They proposed a transformation method from UML state diagrams into flow graphs and showed that conventional flow analysis techniques can be applied to test case generation from UML state diagrams [**7**].

A.V.K. Shanthi et al. presented the test case generation by means of UML Sequence diagram using Genetic Algorithm in which best test cases are optimized. Their approach is significant to identify location of a fault in the implementation, thus reducing testing effort. Moreover this method inspires the developers to improve the design quality, find faults in the implementation early, and reduce software development time [**19**]. L. Wang et al., gave an approach to generate test case from UML activity diagrams based on Gray-Box method. It demonstrates a systematic method to generate test cases directly from UML activity diagrams, and many parts of this method could be automated [**20**].

J. Offutt et al. has introduced new integration-level analysis and testing techniques that are based on design descriptions of software component interactions. There are relatively few formal testing criteria that are based on design descriptions. The techniques in this paper are innovative in that they utilize formal design descriptions as a basis, and have practical value because they can be completely automated from the widely used design notation of collaboration diagrams. Tools already exist for constructing collaboration diagrams, and tools for performing the analysis and testing described here can be built with relative ease. This paper also includes an algorithm for instrumentation of a program that is implemented from collaboration diagrams. The instrumentation will ensure that tests satisfy the formal testing criteria developed in this research, and also help ensure traceability from the design artifacts to the code [**21**].

JA Whittaker described a clearer view of some of software testing's inherent difficulties in his paper, testing can be done in four phases: Modeling the software's environment, Selecting test scenarios, Running and evaluating test scenarios and Measuring testing progress. The first and most important thing to be done is to recognize the complex nature of testing and take it seriously. The author advised to hire the smartest people, help them get the tools and training they need to learn their craft, and listen to them when they tell about the quality of the software [**22**].

J.Offutt et al. introduces a new technique for generating tests from formal software specifications. Formal specifications represent a significant opportunity for testing because they precisely describe the functionality of the software in a form that can be easily manipulated by automated means. This research addresses the problem of developing formalizable, measurable criteria for generating tests from specifications. Results from applying the criteria and process to a small example were presented. This case study was evaluated using Atac to measure decision coverage, and the technique was found to achieve a high level of coverage. It was also used to successfully detect a large percentage of faults. These results indicate that this technique can benefit software developers who construct formal specifications during development [**23**].

Alexander Pretschner et al. presented their continuing efforts in specification based test sequence generation and its embedding in an incremental Software development process. The class of systems is neither restricted to finite nor to deterministic ones: recursive data types or real numbers are handled in exactly the same way as finite enumeration types (with the problem of finding appropriate instantiations); non-determinism is handled by the backtracking mechanisms in CLP (with the problem of properly formulating verdicts). Experience with our industrial partners shows that customized management systems for (regression) testing are at least as important as a systematic generation of test cases; this is, however, not the focus of our current work. In order to assess the scalability of our CLP based approach, we are carrying out an industrial size case study with a large German manufacturer of smart cards; first results give us some reason to be optimistic but show that more intelligence in the process of search is necessary [**24**].

Umar Farooq et al. this research surveys and analyzes empirical studies on evaluation of testing techniques and proposes a uniform classification technique and identify asset of factors which helps in selecting appropriate testing technique. Results indicate that there is no perfect technique and every technique has its weaknesses, but combination of different techniques is more effective solution to increase reliability in Software [**25**].

Saru Dhir had provided a brief explanation of the testing UML 2.0. The UML 2.0 provides the different techniques and tools for generating the test cases. In this paper, the focused on the different techniques used for generating test cases that minimize the number of test cases in behavioral and Structural diagrams. Behavioral elements from UML 2.0 can be used to specify the dynamic nature of test cases. These include interaction, state chart and activity diagrams. Structural elements includes: class diagram, object diagram and implementation diagrams [**26**].

Hajar Homayouni et al. represented two different classification frameworks for the existing automatic test case generation approaches, and also have a brief look at each one. They described how to evaluate generated test cases, and introduce a classification of evaluation approaches. The results show that different approaches should be selected based on types of applications, features of software we want to test, technique's complexity, and other features. Although there have been lots of researches on automatic test case generation problem, but for real world systems more researches are still needed[**27**].

Zhanqi Cui et al. In this paper, an aspect-oriented approach for modeling and integrating crosscutting concerns as sequential and parallel aspect models based on UML activity diagrams is presented. They add lightweight extensions to standard activity diagrams with stereotypes and tag values. An aspect model is designed as pairs of point cut model and advice model with extended activity diagrams. Advice models are woven into primary models according to corresponding point cut models. Two case studies have been conducted to demonstrate the feasibility of our approach. Concerning the future work, they will focus on verifying integrated models against system requirements and testing system implementation against the verified models. They also intend to build an aspect model repository of typical crosscutting concerns. The aspect models can then be reused in different applications [**28**].

Raluca Lefticaru et al. presented an approach for automatic generation of test data, using state diagrams and genetic algorithms. The strategy is simple, the derivation of the fitness function is straightforward and so could be easily adopted in industrial software development. Furthermore, experimental evidence show that the test data obtained can cover difficult paths in the machine and a slightly different design of the fitness function can be used for specification conformance testing [**29**].

3. UNIFIED MODELING LANGUAGE

The Unified Modeling Language (UML) is a collection of languages for specifying, visualizing, constructing, and documenting the artifacts of software systems [**30**]. Here complex systems are designed and modeled through a collection of views of a model. The UML defines nine separate graphical diagrams to specify and design software.

UML has a broad spectrum of usage. It can be used for business modeling, software modeling in all phases of development and for all types of systems, and general modeling of any construction that has both a static structure and dynamic behavior. For achieving these wide-ranging capabilities, the language is defined to be extensive and generic enough to allow for the modeling such diverse systems, avoiding the too specialized and too complex [**31**]. The overview of UML has the following different parts:

Views: Views show different aspects of the system that modeled. It is not a graph, but an abstraction consisting of a number of diagrams. Only by defining a number of views, each showing a particular aspect of the system, can a complete picture of the system be constructed. To link the modeling language to the method/process chosen for development the views are used.

Diagrams: The graphs that describe the contents in a view are called diagrams. UML has nine different diagram types that are used in combination to provide all views of the system.

Model elements: The concepts used in the diagrams are model elements that represent common object-oriented concepts such as classes, objects, and messages, and the relationships among these concepts including association, dependency, and generalization. A model element is used in several different diagrams, but it always has the same meaning and symbol.

General mechanisms: General mechanisms provide extra comments, information, or semantics about a model element; they also provide extension mechanisms to adapt or extend the UML to a specific method/process, organization, or user [**31**].

3.1 UML Based Test Case Generation Methods

A popular approach, use of UML models for test case generation is discussed in [**32**] [**33**] [**34**] [**35**] [**36**] and[**37**]. Most of the approaches of UML-based methods for test case generation are originally developed for automatic generation of test cases, but they also help to think in a way of applying these methods to hardware-based system test case generation and come up with a methodological approach. Some standardized models for the generation of test cases are required for automation of tests. Scenarios and use cases which are the elements of UML do not only feed requirements, but they also build the bases for testing [**35**]. The use case models are transferred to behavioral diagrams, these diagrams are refined according to method specifications and these refined diagrams are used to generate system-level test cases. Test cases for black-box testing that is functional behavior of the system [**32**][**35**][**36**][**37**] are generated through these methods.

3.1.1 Test Case Generation Using State Machines

By considering theirs preconditions, post conditions, extensions and variations use cases are generated. The scenarios that the use cases contain are transferred to state diagrams (or state machines). These state machines have transitions specifying pre and post conditions and message transfers between states and test cases gathered from these state diagrams [**32**]. A similar approach to system testing is advised. Again, use case diagrams and use cases are generated and these diagrams are converted to state diagrams. The state diagrams are converted to a defined diagram named as usage graph. A directed graph with a start node and an end node is a usage graph and usage states between them. Usage states are connected to each other with transitions which are actually user actions. Usage models are used in generating test cases by executing the transitions (user actions) between the usage states. Furthermore, white-box testing issues are discussed which are concentrated on structural behavior of software systems which is out of scope of this thesis work [**35**].

3.1.2 Test Case Generation Using Activity Diagrams

The approach described, tests the system from the user's viewpoint. First, use cases are gathered from requirements, which are then going to be used to build activity diagrams. Then, activity diagrams are converted to interaction flow diagrams (IFD), which can be defined as an intermediate step to generate test plans. IFD, as it can be understood from its name, reduce the activity diagrams by subtracting intermediate steps (for example interaction between system modules) between user interaction steps. At the end, IFDs are converted to interaction flow graphs (IFG) which are based on a tree structure with no loops in order to have distinct scenarios for the corresponding use case which can be also defined as test cases. To ensure that each cycle is executed once [**33**] then this test tree is executed based on a Depth-First- Search algorithm.

3.1.3 Test Case Generation Using Sequence Diagrams

The approaches having use UML sequence diagrams, which include the information of interaction of system with actors, in order to generate test cases. The approach defines a special diagram gathered from sequence diagram to clearly define the scenario paths and uses the diagram to cover the user-system interactions. The approaches in [18] and [19] includes the black-box testing and white-box testing issues together, considering the structural behavior and functional behavior of the system under test and does not advice a method to reduce the whole testing issue to a functional system testing process[**36**][**37**].

3.1.4 Test Case Generation Using Use Cases

For test case generation from use cases another approach is discussed here. The approach is based on the rule for test cases that each scenario or instance of a use case should correspond to a test case and this approach brings the advantage of preventing the consequences of incomplete, incorrect and missing test cases as other approaches also provide. The approach offers first building a system boundary diagram depicting the interfaces between the software being tested and the individuals, systems and other interfaces; secondly use cases are generated for all actors defined in system boundary diagram. At the end, test cases are generated in a way that there exist at least two test cases for one use case which are successful execution of test case and unsuccessful execution of test case. Clearly, much more test cases can be generated for a use case for exceptions and alternative courses [**34**].

Furthermore, there are also approaches for functional testing for SPL. A set of software intensive systems which shares a common and managed set of features satisfying the specific needs of a particular market segment or mission is called an SPL.A large number of studies has been done for SPL testing which are covering all the testing levels in the lifecycle of software and unit testing to functional testing. The most important ones of those works about this thesis work's scope are functional testing approaches. A lot of methods are discussed, but mostly studies have been done using UML diagrams which have already been discussed above [**38**].

4. METHODOLOGY FOR TEST CASE GENERATION

The method for generating test cases from different diagrams, include number of activities. This is a process. The main activities are shown in Fig 1.

4.1 Construction of Intermediate Model

Several strategies have been reported to generate test cases using a variety of models. However, in many cases the test cases based on more than one model type. In such cases, it becomes necessary to first construct an integrated model based on the information present in different models.

4.2 Generation of Test Scenarios

The test cases which are generated from models are present in the form of sequences of test scenarios. Test scenarios specify a high level test case rather than the exact data to be input to the system. For example, in the case of FSMs, it can be the sequence in which specifies states and transitions must be undertaken to test the system-called a transition path. The sequences of different transition labels along the generated paths form the required test scenarios. Similarly the message paths can be generated from the sequence diagrams. Also shown are the exact sequence messages in which the classes must interact for testing the system.

4.3 Test Generation

The difficulty of generating tests from a model depends on the nature of the model. Models that are useful for testing usually possess properties making test generation effortless and automatable frequently. For some models, all that is required is to go through combinations of conditions described in the model, which require simple knowledge of combinatory. There are a variety of constraints on what constitutes a path to meet the criteria for tests. It includes having the path start and end in the starting state, restricting the number of loops or cycles in a path, and restricting the states that a path can visit.

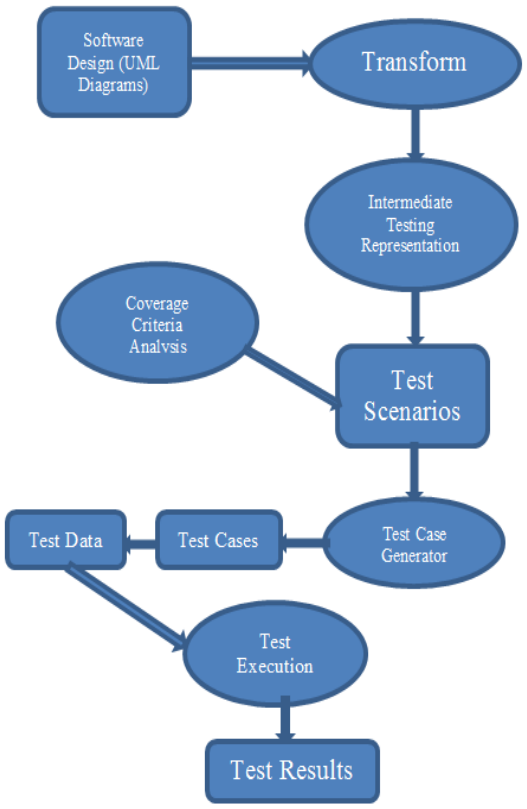
Software Design (UML Diagrams)
Transform
Intermediate Testing Representation
Coverage Criteria Analysis
Test Scenarios
Test Data
Test Cases
Test Case Generator
Test Execution
Test Results

Fig. 1 Activities of Test Case Generation

4.4 Automatic Test Case Execution

In certain cases the tests can even be performed manually. Manual testing is labor-intensive and time consuming. However, for manual execution the generated test suite is usually too large. Moreover, a key point in UML based technique is the frequent regeneration and re-running of the test suite whenever the underlying model is changed. Accordingly for achieving the full potential, automated test execution is required. Usually by using the available testing interface for the software, the abstract test suite is translated into an executable test script. Automatic test case execution also involves test coverage analysis so that the tests generation step may be fine-tuned or different strategies may be tried out.

4.5 Test Coverage Analysis

Each test generation method targets certain specific features of the system to be tested. Using test coverage analysis the extent to which the targeted features are tested can be determined using test coverage analysis. The important coverage analysis based on a model can be the following: all model parts (or test scenarios) coverage is achieved when at least once the test reaches every part in the model.

Important test coverage required based on UML models can be the following: path coverage, message path coverage, transition path coverage, scenario coverage, dataflow coverage, polymorphic coverage, inheritance coverage. When the test executes every scenario identifiable in the model at least once then scenarios coverage is achieved.

5. CONCLUSION

In this paper, an explanation of testing techniques based on UML has been provided. The UML provides different techniques and tools for generating the test cases. In this paper, the focus is also on the different techniques used for generating test cases that can minimize the number of test cases. The study shows that UML can be used to specify the dynamic and static nature of test cases. These include interaction, state-chart, use case, class and activity diagrams.
In future, there is a plan to develop a new technique for generating test cases from UML.

6. REFERENCES

[1] R. Pressman, *Software Engineering: A Practitioner's Approach*.: Mc-GrawHill, 2005.

[2] tutorialpoint. [Online]. www.tutorialspoint.com/software_testing

[3] Karambir et al, "survey of sotware test case generation techniques," *International journal of advanced research in computer science and software engineering*, 2013.

[4] I Sommerville, *Software Engineering*. England: Addison-Wesley, 2000.

[5] Prasanna, M. et al, "A survey on automatic test case generation," *Acad. open Internet J.*, 2005.

[6] Jirapun Daengdej, "A Test Case Generation Process And technique," *Academic Journals Inc.*, 2010.

[7] H.S Hong, Y.G Kim,S.M Cho, D.H Bae, S.D Cha, "Test Case Generation from UML state Diagrams," , Korea, 1999.

[8] Neelam Gupta, AdityaP.Mathur,Marry lon Soffa, "Generating Test Data for Branch Coverage," , Pittsburgh, 2000.

[9] F. Basanieri, A. Bertomated ,E. Marchetti, A. Rinoline, G. Lombardi, "An Automated Test Strategy Based on UML Diagrams," , Sweden, 2001.

[10] M. Sarma, R. Mall, "Automatic Test Case Generation from UML Models," , 2007.

[11] Chen Mingsong, Qiu Xiaokang, and Li Xuandong, "Automatic Test Case Generation for UML Activity Diagrams".

[12] Debasish Kundu, Debasis Samanta, "A Novel Approach to Generate Test Cases from UML Activity Diagrams," vol. 8, no. 3, 2009.

[13] R. Mall et al., "Test Case Generation Based on Use case and Sequence Diagram".

[14] Prasanna M., Chandran K.R, "Automatic Test Case Generation for UML Object Diagrams using Genetic Algorithm ," vol. 1, 2009.

[15] Jeff Offutt, Aynur Abdurazik, Andrea Baldini,Supaporn Kansomkeat, "A Comparative Evaluation of Tests Generated from Different UML Diagrams".

[16] Santosh Kumar Swain, Durga Prasad Mohapatra, Rajib Mall, "Test Case Generation Based on State and Activity Models," 2010.

[17] Philip Samuel, Rajib Mall, "A Novel Test Case Design Technique Using Dynamic Slicing of UML Sequence Diagrams," vol. 2, no. 1, 2008.

[18] Yiwen Wang, Mao Zheng, "Test Case Generation from UML Models".

[19] A.V.K. Shanthi, G. Mohan Kumar, "Automated Test Cases Generation from UML Sequence Diagram," , Singapore, 2012.

[20] L. Wang, J. Yuan, X. Yu, J. Hu, X. Li, and G. Zheng, "Generating Test Cases from UML Activity Diagram," , 2004.

[21] J. Offutt, A. Abdurazik, "Using UML Collaboration Diagrams for Static Checking and Test Generation," , USA, 2000.

[22] James A Wittaker, "What Is Software Testing? And Why Is It So Hard?," vol. 17, no. 1, 2000.

[23] J.Offutt,Y.Xiong,S.Liu, "Criteria for Generating Specification based Tests".

[24] Alexander Pretschner, Heiko L¨otzbeyer, Jan Philipps, "Model Based Testing in Evolutionary Software Development," 2001.

[25] U. Farooq, Evaluating Effectiveness of Software Testing Techniques with Emphasis on Enhancing Software Reliability , 2012.

[26] Saru dhir, "IMPACT OF UML TECHNIQUES IN TEST CASE GENERATION," vol. 2, no. 2, 2013.

[27] Mohammad Reza Keyvanpour, Hajar Homayouni, Hossein Shirazee, "Automatic Software Test Case Generation: An Analytical Classification Framework," vol. 6, no. 4, 2012.

[28] Zhanqi Cui, Linzhang Wang, Xuandong Li, "Modeling and Integrating Aspects with UML Activity Diagrams".

[29] Raluca Lefticaru, Florentin Ipate, "Automatic State-Based Test Generation Using Genetic Algorithms".

[30] (1999, June) Object Management Group. [Online]. www.omg.org/uml/

[31] Jim A., Ila N., *UML 2 AND THE UNIFIED PROCESS*.: Pearson, 2006.

[32] Fröhlich, P., Link, J., "Automated Test Cases Generation from Dynamic Models," , Berlin, 2000.

[33] Heinecke, A., Brückmann, T., Griebe, T., Gruhn, V., "Generating Test Plans for Acceptance Tests from UML Activity Diagrams," , 2010.

[34] W.E. Perry, *Effective Methods for Software Testing*.: Wiley, 2006.

[35] Riebisch, M., Philippow, I., Götze, M., "UML-Based Statistical Test Case Generation," , Berlin, 2003.

[36] Sarma, M., Kundu, D., Mall, R., "Automatic Test Case Generation from UML Sequence Diagrams," , 2007.

[37] Sarma, M., Mall, R., "System Testing using UML Models," , 2007.

[38] Lamancha, B.P., Usaola, M.P., Velthius, M.P, "Software Product Line Testing: A Systematic Review," , 2009.