ISSN:0975-9646

Improving Model Based Testing Using Event Consideration For Various Designs Concepts

Nidhi Pare, Prof. Piyush Soni

Department of Computer Science and Engineering
Sanghavi Institute of Management & Science Indore(M.P)-452001 , India

***Abstract*— Software Testing is a major and complex process in software development life cycle. Lots of test cases may be produced to ensure the validity of work, for reduction in test cases, we must focus on its problems. Automation of this test case generation may lead to overcome the above problems slightly and also reduces the human effort in other ways it also helps in detecting the human intended errors and logical errors as well.**

But automation of testing will not be that much productive in terms of time consuming and cost, if we have to wait till the end of the SDLC stage i.e. if we follow the white box testing methodology of testing. we have to go for that part of the code and design document as well, If any errors will be detected in this stage, So we are trying to focus our study on Model Based testing(MBT) approach for both test case generation and test case optimization to achieve some of the goal.

The Main goal of this research is to an improved model based testing tool using event consideration for various designs concepts that can help to provide assurances of reduced overhead of testing. Some techniques for providing such assurances have been developed already, but no single technique has provided a complete solution to the problem. Thus, this thesis will explore the effectiveness of combining two such techniques (Unified Modelling & Combinatorial Testing) into a single tool. The more general purpose of this research is to improve the available methods of software testing. So our work suggests a novel event model based testing mechanism using certain design primitives. The approach aims towards making the efforts reduced for testers end. Testing always works with the later phases of the SDLC which leads towards delayed faults and bug detection.

***Keywords*— Software ,Testing, Model Based Testing(MBT),Test Cases, Unified Modelling & Combinatorial Testing, Software development life cycle(SDLC).**

I. INTRODUCTION

Software is everywhere. An average company spends about 4 to 5 percent of its revenue on Information Technology (IT), whereas companies which are highly IT dependent, such as finance and telecommunications are spending more than 10 percent on it. In other words, IT sector has now one of the largest corporate expenses, outside employee costs. A lot of that money goes into hardware and software upgrades, software license fees, but a big chunk is for new software projects meant to create a better future for an organization and its customers. Software projects are inherently complex, risky and require careful planning. Proper planning ensures that a project doesn't fail, while at the same time, customers get a clear definition of the project, know the project status and have a ready access to project deliverables at any point of time. Most recent surveys have shown that inadequate planning and specifications, ill defined requirements, poor process of requirement analysis and testing of the system, short comings of metrics and measures to compute project's sheer size and complexity level, overall lead to numerous change requests, delays, significant added costs and increase in the possibility of errors. Over 41% of the IT development budget for software, staff and external professional services is consumed by poor requirements at an average, where the company is using average analysts. Sloppy development practices are also a rich source of failure and they can cause errors at any stage of an IT project. Moreover, the costs of errors that are introduced during requirements phase and fixed later in the Software Development Life Cycle (SDLC) increase exponentially.

II. BACKGROUND

- **Software Testing**

Software testing is the process of exercising a program with well designed input data with the intent of watching disappointments. As it were, Testing is the process of executing a program with the expectation of discovering errors [1]. Testing distinguishes faults, whose evacuation builds the software quality by expanding the software potential reliability. Testing additionally measures the software quality as far as its ability for accomplishing correctness, reliability, usability, maintainability, reusability and testability. The various Objectives of testing are as follows:

- Testing should also aim at suggesting changes or modifications if required, thus adding value to the entire process.
- The mail goals is to design and creates tests that systematically uncover different classes of errors and do so with a minimum amount of time and effort.
- Performance requirements are required as it specified in specification document.
- Software reliability and software quality based on the data collected during testing.

The various advantages of testing are as follows:

- Increasing accountability and Control
- Cost reduction
- Time reduction
- Defect reduction
- Productivity enhancements of the Software developers

Test case generation from design specifications has the added advantage of allowing test cases to be

available early in the software development cycle, thereby making test planning more effective [2].

- **Importance of Model Based Testing (MBT)**

Testing methodologies which uses model is called model based testing (MBT). Model based testing (MBT) refers to the type of process that focuses on deriving a test model using different types of formal ones, then converting this test model into a concrete set of test cases [5]. Models are the intermediate artifacts between requirement specification and final code. Models preserve the essential information from the requirement, and are the basis for implementation. Instrumentation of models into testing process is the prime subject of concern of our thesis. Development of unified modelling language (UML) has helped a lot to visualize/realize the software development process. At the earliest stage of software development life cycle (SDLC), no one including user and developer can see the software; only at the final stage of the product development it is possible. Any errors/problems found out at the final stage, it incurs a lot of cost and time to rectify, which is very much crucial in IT industry.

UML is the modelling language, which supports object-oriented features at the core. In the last few years, object-oriented analysis and design (OOAD) has come into existence, it has found widespread acceptance in the industry as well as in academics. We concentrate here on widely accepted practices based on the use of the Unified Modelling Language (UML) to support an object-oriented development process [6]. The main reason for the popularity of OOAD is that it holds the following promises:

• Code and design reuse
• Increased productivity
• Ease of testing and maintenance
• Better code and design understanding ability

UML accomplish the visualization of software at early stage of SDLC, which helps in many ways like confidence of both developer and the end user on the system, earlier error detection through proper analysis of design and etc. UML also helps in making the proper documentation of the software and so maintains the consistency in between the specification and design document. The key advantage of this technique is that the test generation can systematically derive all combination of tests associated with the requirements represented in the model to automate both the test design and test execution process.

- **Consequences of Poor Testing**

According to Gartner Research, "The lack of testing and Quality Assurance (QA) standards, as well as a lack of consistency, often lead to software failure and business disruption, which can be costly." Results of a survey conducted in June 2010 have also shown that majority of software bugs are attributed to poor testing procedures or infrastructure limitations, rather than design problems. A report that has cited six common failures of IT projects has shown that poor testing and absence of proper change management are two main reasons of software failure.

- **Event Driven Testing- new approach**

Events have also played a very important role to model object interaction, develop a metric suite for domain modelling and for analogical reuse of structural and behavioural aspects of event-based object-oriented domain models. Events are modeled in terms of object-oriented structures, like entities and as four different events in UML language. Events are also used to model information structures and related activities in information systems and for modelling static and dynamic aspects of Information and Communication Systems.

III. LITERATURE REVIEW

Software testing relates to the process of finding defects and ensuring that particular software of interest meets its specification. One of the key activities within software testing is on the test case design. Over the years, many test case design strategies have been developed in the literature including that of combinatorial testing, boundary values, equivalence partitioning, decision tables, robustness consideration and in addition cost and effect graphing. Although helpful, these strategies don't sufficiently cater for bugs because of interaction. Tending to the aforementioned issue, numerous researches into interaction based strategies, called t-way strategies (where t represents interaction strength), have started to develop in the literature. Model based testing (MBT) is one of the most important testing strategy from which early test case design is possible. MBT uses UML models which are widely used to describe analysis and design specifications of software development. UML 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.

The paper [7] proposes a novel method to automatically generate test cases based on UML Class diagrams .In this paper author gives a procedure to extract data from UML class diagram & mapped it to genetic algorithms for combinatorial outputs. Then it generates a tree structure which applies the value is to depth first search to generate all possible valid test cases. Efficiency of test cases can be analyzed with help of mutation testing. It has four step procedure tree formation, new trees formation, test cases generation & test cases evaluation. From the proposed approach of specification-based testing the complexity is reduced too much extent. It also uses information derived from a specification to assist testing as well as to develop program.

This paper [8] explores different approaches that had emerged during the past decade regarding the generation of test cases and test data from different models as an emerging type of model based testing. It gives the brief comparison for different techniques. Unified Modelling Language UML models took the greatest share from among those models. All the techniques first categorize the UML model in three basic types of diagram behavioral, interactional, structural diagrams. After the categorization of UML diagrams author yields to a categorization of the test cases generation techniques according to the diagram(s) being used. Thus paper mainly focuses on test data extraction technique based on meta-heuristic search phenomenon's. Techniques such as genetic algorithms, behavioral regression testing (BERT),

metamorphic & extended finite state machine (EFSM) can be used for optimizing the test case generation complexity.

In this scheme of scenario-based testing [9], test scenarios are used for generating test cases, test drivers etc. In this paper author generate test scenarios from sequence & class diagrams, which achieve test adequacy criteria perfectly which achieves maximum path coverage criteria. Author proposes an approach TC-ASEC to generate test cases from design models using sequence diagram and class diagram. After that we analyse the sequence diagram to find the interaction categories and then use the class diagrams to find the settings categories. When all the scenarios is generated they uses category partitioning method to analyse functional requirements & divide the in various functional units. For each defined functional unit, conditions in the environment (system characteristic of a certain functional unit) and the parameters (explicit input of the same unit) relevant for testing must be found. Further test cases are then derived by finding significant values of environment conditions and parameters. Moreover the overall approach is not fully automated. Another automated tool can be developed for the proposed approach. The ultimate goal will be to address testability, coverage criteria and automation anomalies, in a way to fully support system testing activities.

In this paper a new approach is proposed for combinatorial testing by applying test prioritization in event-driven software (EDS) [10]. All EDS take sequences of events (e.g., messages, mouse-clicks) as input, change their state, and produce an output (e.g., events, system calls, text messages). An additional criterion is developed to prioritize GUI and web-based programs known as prioritizing function (PF). The PF takes as input a set of test cases to be ordered, and returns a sequence that is ordered by the prioritization criterion & consider it as parameter. It also uses a function (called *Order Suite*) which selects a test case that covers the maximum number of criteria elements (e.g., windows, parameters) not yet covered by already-selected test cases. The function iterates until all test cases have been ordered. In half of these experiments, event interaction- based prioritization results in the fastest fault detection density (FDD), where FDD is a measure of the number of faults that each test identifies on average.

It is the model based test suite generation process which proposes an integrated approach to generate test cases from UML sequence and activity diagrams [11]. This approach transforms these UML diagrams into intermediate representation message flow graph (MFG), after that an algorithm is used to generate test scenarios from the constructed graph. Now, the necessary information for test case generation, such as method-activity sequence diagrams, associated objects in the current context, and constraint conditions are extracted from test scenario. The test sequences are a set of theoretical paths starting from initialization to end, while taking conditions (pre-condition and post-condition) into consideration. Each generated test sequence corresponds to a particular scenario of the considered use case of the system. The third phase is to generate test case from the generated sequences satisfying the message-activity path *test adequacy criteria*. This approach reduces the number of test cases and still achieves adequate test coverage of the system. Now, achieving message-activity path coverage and category partitioning method for each predicate conditions found in the specific path of the design model.

This paper [12] presents a technique that enhances the integration testing of classes by accounting for all possible states of interacting objects. The proposed approach SATEC (State Activity Test Case generation) is categorized in four parts: SAD generation, basis path generation (using coverage criteria), test scenario generation, test case generation. In SAD (state-activity-diagram), 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. The purpose is to handle concurrent execution of the code and some new types of nodes have been introduced. Several faults such as incorrect actions to an event, correct event passed to a wrong object or incorrect events passed to the right object in its correct state, incorrect method invocation in an activity in the system, sneak transitions, incorrect or missing output, etc. may occur in an operation of the process. The results to an event depends upon the corresponding object's state. A test set is therefore necessary to detect faults if any when an object invokes a method of another object and whether the right sequence of states and activities is followed to accomplish an operation.

ITTDG in [13] implements the greedy method in deciding which test data will be selected as the final test data. The strategy iterates all uncovered tuples produced by every interaction or input-output relationship specify by the user. Iteratively, the strategy will push the visited tuples into a list referred as test data candidates list. The test data candidates list then will be extended by adding one parameter at a time with value that covers the most uncovered tuples. In case of "tie" situation (that is, more than one value cover the most uncovered tuples), the corresponding test data will be duplicated with all the tie values and all duplicated test data will be pushed into Q. Once all test data in Q form a complete test data, the test data which has the highest weight (that is, covers the most uncovered tuples) among the test data candidates in Q will be selected as the final test data. The selected test data then will be pushed into the final test suite and the tuples covered by the test data are removed from uncovered tuples list. As the size of Q can potentially grow significantly during the parameter extension process, the number of test data candidates in Q subjected to a constant integer value (M) to avoid the possible out-of-memory error (that is, when dealing with large number of parameters and values). It provides seamless integration of all interaction possibilities. It gives better performance as far as the generated test size is concerned especially involving uniform number of parameter values.

According to the paper [14], author directs towards early generation of test cases from various design

phases for Concurrent Systems using UML sequence diagram. For this the author takes sequence diagram as input and mapped it to an intermediate concurrent composite graph (CCG). For extracting the value of input the graph is been completely traversed by breadth first search (BFS) and depth first search (DFS) technique. Using message sequence path criteria to generate the test cases for concurrent systems a scenario diagram (SD) combines multiple scenarios by a combined fragment (CF) function. After getting all the above values an algorithm is proposed which generate message sequence path criteria (MSPC) which automatically traverse the concurrent composite graph and identified the valid parameters & values.

Review Extraction-

The problem identified with this approach is that multiple object accessing the same function at same time cannot be handled .It also not gives any of the priority to user for high priority test case. Also for the above described method construction of composite graph is necessary which seems to restrict us and also increases an overhead. In our research we are taking the key concern from both the author and finds that the early generation of test cases will reduces the test cost & size. By applying all the above methodology in a proper manner we achieved an excellent result. Empirical study shows that for pairwise test generation we achieved 100% coverage criteria and further more results on various parameters is discussed in later section of thesis. Thus, by considering the entire above research objective we have developed a Automated Tool which follow UML modelling based testing.

IV. PROBLEM IDENTIFICATION

Software Testing is a time consuming and costly process in software development life cycle. For reduction in test cases though MBT testing we must focus on its problem domain which in NP Complete so solution must identified while keeping this in mind. Instead of that we are also facing the problem for Automation of test case design process which can result in significant reductions in time and effort, and at the same time it can help in achieving an increased reliability of the software through increased test coverage. Automation of this phase may lead to overcome the above problems and also reduces the human effort in other ways it also helps in detecting the human intended errors and logical errors as well. Automation of testing will not be that much productive in terms of time consuming and cost, if we have to wait till the end of the SDLC stage i.e. if we follow the white box testing methodology of testing. If any errors will be detected in this stage, we have to go for that part of the code and design document as well. We have to follow up strict verification of both code and design document from beginning to short out the error. So only one solution to this problem is to, start the testing process from early stage of SDLC i.e. from requirement specification stage through design phase up to the last phase. So we focused our study on Model Based testing approach for both test case generation and test case optimization to achieve some of the goal, described below:

- To propose some generalized techniques to generate test cases for object-oriented software's using UML Activity Diagram diagrams.
- To propose a generalized technique for optimized test case generation using UML diagrams.
- To do implementation of the proposed methods and evaluate their effectiveness.
-

V. RAITONALE OF WORK

The immediate purpose of this research is to an improved model based testing tool using event consideration for various designs concepts that can help to provide assurances of reduced overhead of testing. Some techniques for providing such assurances have been developed already, but no single technique has provided a complete solution to the problem. Thus, this thesis will explore the effectiveness of combining two such techniques (Unified Modelling & Combinatorial Testing) into a single tool. The more general purpose of this research is to improve the available methods of software testing. There are several major challenges that completely resolved by our tool with testing modern software. Some are as follows:

- Fully Automated test case generation and execution.
- Formalization and modelling of the software specifications and implementations, and software testing process and effects. The reduction in growing complexity of the modern software-based systems.
- Ability to Generate Test Cases Criteria at the Time of Design and Requirement Analysis (Early Test Case Creation saves time & cost).
- Extraction the data to generate test cases, from UML diagram. Generating the reduced number of test cases (Test Suite Size).
- Providing the maximum test coverage (100%) with reduced execution time for testing.

VI. PROPOSED SOLUTION

This work suggests a novel event model based testing mechanism using certain design primitives. The approach aims towards making the efforts reduced for testers end. Testing always works with the later phases of the SDLC which leads towards delayed faults and bug detection. This works also aims towards making this detection in early phases of development which leads towards improving the designing and coding part of the software products. The quality of the software is analysed using various attributes and testing forces to be one of the best evaluator. After the NP complete problem is been solved completely, the generation of reduced pair sets is feasible but to show the result of research we have to develop a prototype for the domain. Also our one of the main objective is early test case generation to reduce cost & efforts. So to accomplish above mentioned goals this work proposes new design architecture of event model based testing mechanism. Implementation of suggested model makes it possible to automate the testing procedure through a abstract formal model.

The key concern will be on determining which combination of UML diagrams, and their associated constraints, may be used to automatically, or semi-automatically, generate test cases. It has its two main aspects, First aspect relates to the testable information contained in a UML model, while the Second aspect relates to the development of a technique to generate a test suite from the acquired diagram data. The two major aspects to the proposed study will be explored in five phase process. Initially, as part of the first phase, we must determine what information is been collected by requirement gathering phase then select a proper design model for test case extraction. Once taxonomy of this generic information is established, we must determine which diagrams can provide the necessary information.

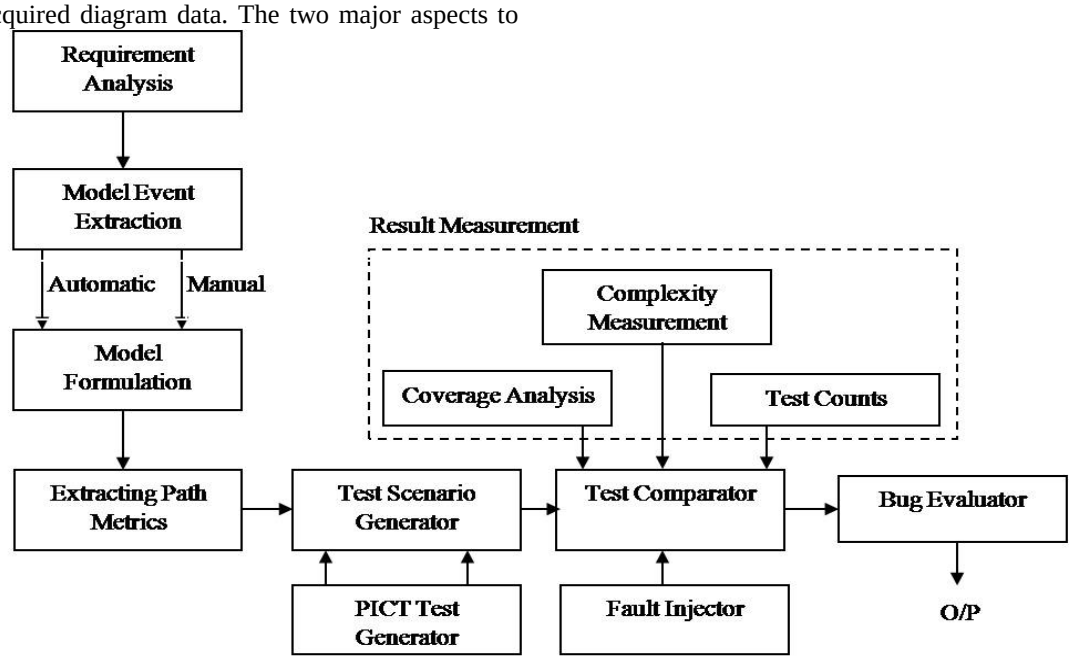

Figure 1: Proposed Design Architecture of Model Based Testing Using Event Consideration

The systems start operating by identifying the testable information or the requirement specification values as results. further process after getting the input it initializes the test set with an empty set & then assigns weights to them. Once the RA information is clear then it is categorized according to the model requirements and specification. In our approach system the approach is using UML modelling language for determining the values and results. Now, this model based values are measured using the event counts. It shows the variation in the current state of the system. If the system is varying with its state continuously means it is having large number of the interaction.

After the interaction event is detected from the system then in calculates the complete path metrics for specifically designed UML diagram such as activity or the sequence. Path metrics calculation is having a benefit that it shows the complete flow of the data along with the each path of the software systems. These extracted statics about the system is used for generating the test scenarios using the PICT combinatorial method of generation of test cases. When the test sets or scenarios are generated then it cloud be evaluated using various comparing factors. Normally for the compete test most of the researchers are taken the coverage criteria as their main factor of evaluation. Also with the coverage factor the complexity issues and the test counts are some other important factors. From this method one by one it identified each test set with most unused counts. Then we will explore which techniques might be suitable for the test data extraction process. Once all the information is been gathered we evaluate the result through injecting some fault in software & analyse the suggested approach for these detection of faults. At the end, now evaluation our technique for effectiveness and efficiency, against other random test case generation methodologies.

Perform Tests: Generated concrete tests are typically executed within a standard automated test execution environment, which is been developed already. Moreover, it is possible to execute tests manually – i.e. a tester runs each generated test on the SUT, records the test execution results and outputs, and compares them against the generated expected outputs. Or take another path, when the tests are executed on the SUT, we find that some tests pass and some tests fail. The failing tests indicate a discrepancy between the SUT and model of the system, offcourse need to be investigated to decide whether the failure is caused by a bug in the SUT.

Result: This step analyses the real system which is to be tested and accepted by user accordingly. The effectiveness of test cases can be evaluated using a fault injection technique called mutation analysis of the system. Mutation testing is a process through which faults are injected into the system to verify the efficiency of the test cases.

VII. EXPECTED BENEFITS

Proposed ASMBT approach have got several benefits over other UML based combinatorial approach is an innovative and high-value generating approach compared to more traditional functional testing methods. Expected key benefits of ASMBT are listed below:

- Model based test case generations are more efficient approach to find out the "loop holes" and remove it
- No dependence from the test execution robot, which is tending to find out the test cases.
- Test case generation is an automated process using this approach;
- Accurate coverage of functional behaviour of the system so to get the higher level of accuracy;
- Continuous maintenance of the requirement coverage matrix;
- Well defined action words explained in UML model operations of the system;
- Well defined test scripts, useful for further propose;

VIII. CONCLUSION

The thought of UML-model based testing utilizing event commitment is to utilize an express abstract model of a SUT and its environment to consequently infer tests for the SUT: the behaviour of the model of the SUT is translated as the expected behaviour of the SUT. The technology of proposed testing has developed to the point where large-scale deployments of this technology are getting to be ordinary. The requirements for achievement, for example, capability of the test team, integrated tool chain availability and routines, are currently recognized, and an extensive variety of commercial and open-source tools are accessible. Despite the fact that it won't tackle all testing issues, it is a vital and helpful strategy, which brings critical advance over the condition of the practice for functional software testing viability, and can build productivity and enhance functional coverage.

REFERENCES

[1] Olli-Pekka Puolitaival "Model based testing tools" published in journal of VTT Technical Research Centre Of Finland

[2] Mark Utting "Position Paper: Model-Based Testing "http://secure.ucd.ie/products/opensource/ESCJava2.

[3] Ibrahim K. El-Far and James A. Whittaker" Model-based Software Testing " Encyclopedia on Software Engineering (edited by J.J. Marciniak), Wiley, 2011

[4] Ren´ee C. Bryce, Ajitha Rajan, Mats P.E. Heimdahl " Interaction Testing in Model-Based Development: Effect on Model-Coverage" ASIA PACIFIC SOFTWARE ENGINEERING CONFERENCE (APSEC'06)

[5] H. Y. Ong and Kamal Z. Zamli " Development of interaction test suite generation strategy with input- output mapping supports" Scientific Research and Essays Vol. 6(16), pp. 3418-3430, 19 August, 2011

[6] Cle´mentine Nebut, Franck Fleurey, Yves Le Traon, Member, IEEE, and Jean-Marc Je´ze´ quel, Member, IEEE "Automatic Test Generation: A Use Case Driven Approach " IEEE Transactions On Software Engineering, Vol. 32, No. 3, March 2006

[7] A.V.K.Shanthi, Dr.G.Mohankumar" Automated Test Cases Generation For Object Oriented Software " Indian Journal Of Computer Science And Engineering (Ijcse) 2011.

[8] Pakinam N. Boghdady, Nagwa L. Badr, Mohamed Hashem and Mohamed F.Tolba " Test Case Generation and Test Data Extraction Techniques " International Journal of Electrical & Computer Sciences IJECS-IJENS Vol: 11 No: 03 2011

[9] Baikuntha Narayan Biswal, Pragyan Nanda, Durga Prasad Mohapatra" A Novel Approach for Scenario-Based Test Case Generation " International Conference on Information Technology IEEE 2008

[10] Ren´ee C Bryce, Member, IEEE, Sreedevi Sampath, Member, IEEE, and Atif M Memon, Member, IEEE " Developing a Single Model and Test Prioritization Strategies for Event-Driven Software " published in IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 37, NO. 1, JAN/FEB 2011

[11] Santosh Kumar Swain, Durga Prasad Mohapatra" Test Case Generation from Behavioral UML Models" International Journal of Computer Applications (0975 – 8887) Volume 6– No.8, September 2010

[12] Santosh Kumar Swain, Durga Prasad Mohapatra, Rajib Mall" Test Case Generation Based on State
and Activity Models " Journal of Object Technology Published by ETH Zurich, Chair of Software Engineering, ©JOT 2010

[13] Rozmie R. Othman1 and Kamal Z. Zamli " ITTDG: Integrated T-way test data generation strategy for interaction testing " Scientific Research and Essays Vol. 6(17), pp. 3638-3648, 26 August, 2011
Available online at http://www.academicjournals.org/SRE ISSN 1992-2248 ©2011 Academic Journals

[14] Monalisha Khandai, Arup Abhinna Acharya, Durga Prasad Mohapatra" A Novel Approach of Test Case Generation for Concurrent Systems Using UML " IEEE 2011

[15] Elder M. Rodrigues, Flavio M. Oliveira, Maicon Bernardino " Evaluating Capture and Replay and Model-based Performance Testing Tools: An Empirical Comparison " ACM 2014[Base1]

[16] Florian Häser, Michael Felderer, Ruth Breu " Software Paradigms, Assessment Types and Non-Functional Requirements in Model-Based Integration Testing: A Systematic Literature Review " ACM 2014[Base 2]