International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459 (Online), An ISO 9001:2008 Certified Journal, Volume 3, Special Issue 1, January 2013)

International Conference on Information Systems and Computing (ICISC-2013), INDIA.

TEST CASE GENERATION FROM UML MODELS – A SURVEY

S. Shanmuga Priya[1], Dr. P. D. Sheba[2]

[1]*PG Scholar,* [2]*Professor, Department of CSE, J.J. College of Engineering and Technology, Tiruchirappalli, India*

Email: priya.soundararajan@gmail.com, pdsheba@yahoo.co.in

Abstract

Studies show that large software projects have a huge likelihood of failure. In fact, it's more likely that a large software application fails to meet its requirements on time and on budget due to lack of quality in design or implementation or testing. The success of such software, however, can be improved by modeling. Model-Based test case generation approaches inspire developer to improve design quality by identifying faults in the implementation at early stage, and thereby reducing the software development time. Modeling is one way to visualize the design and check it against requirements before developers begin the coding phase. One such modeling language widely used is Unified Modeling Language (UML). UML diagrams produce a graphical representation (view/model) of the system under design and implementation. If testing the software is addressed at the initial stage in Software Development Life Cycle (SDLC), it will be easy for the testers to test the software in the later stages leading to lesser time consumption as well as lessen the manpower involved. Developing test cases is an essential testing artifact which requires plan, techniques and practice. When testing software, there are often a massive amount of possible test-cases even in quite simple systems. Running all the possible test-cases is almost never an option, and designing test-cases becomes an important part of the testing process. The number of test cases to develop is not limited, but the fewest test cases with maximum coverage should be the goal. Test cases should be designed in such a way that achieves 100% requirement coverage and should be effective in uncovering defects. This paper aims at bringing into limelight the various UML models used for generating test cases.

Keywords-- **UML diagram, Class diagram, State chart diagram, Sequence diagram, Activity diagram, Collaboration diagram.**

I. INTRODUCTION

Many methods that have been used for generating automated test cases are from formal methods, semi-formal methods or an integrated method. Formal Methods are mathematical-based languages, techniques and tools. They are able to be proved and verified as they are built based on the behavioural/structural properties of the System Under Test (SUT). This method has the potential to verify whether the system has been implemented correctly, discover the inconsistencies, and ambiguities. But the major downside of formal method oriented test case generation are i) insufficient tool support, ii) lack of expertise/training, and iii) specifications are easy to understand, but difficult to create. Semi-Formal methods are model-driven engineering that focus on creating models of a system at each stage in the development life cycle. With a well defined methodology automatic model transformations (e.g. to code) from the design level specifications or requirement level specifications are achievable. The major advantages of this method are they are intuitive, and produce more maintainable software.

This method is widely used in software industries. Sometimes an integrated method is followed by taking the best from the formal and semi-formal methods which results in speed, intuitive design. There are various techniques used in automatic test case generation which includes SBST (Search Based Software Test case generation), FSM (Finite state Machine), MBT (Model Based Testing) etc., This paper focuses on UML based techniques, the key techniques found in the literature are: UML diagrams, State chart diagrams, sequence diagrams, activity diagrams, class diagrams, collaboration diagrams and a combinational approach of them.

This paper gives a view on the various techniques available to generate the test cases using structure diagrams, behavior diagrams and combinational approaches of the two diagrams. The organization of this paper is Section 2 gives insight on Unified Modeling Language, and Section 3 gives the literature review, Section 4 gives overview of UML modeling tools and Section 5 gives conclusion.

II. UNIFIED MODELING LANGUAGE

The Unified Modeling Language (UML) is a semi-formal modeling language for specifying, constructing and documenting system. Rational Software Corporation and three methodologists in the Information System and Technology Industry, Grady Booch, Ivar Jacobson and James Rumbaugh originally conceived the UML in 1970. UML is the designing of a software application prior to coding and is an important part of large software projects, and helpful to medium and even small projects as well. A model plays the similar role in software development that blueprints and other plans play in the construction of a building.

By using a model, those responsible for software development project's success can guarantee themselves that business functionality is complete and correct, end-user needs are met, and program, design, support, requirements for scalability, robustness, security, extendibility, and other characteristics, before implementation of the code renders changes complicated and costly to make.

Evolution of UML

UML diagrams are widely gaining its popularity and are used for designing and modeling. The various versions (in the order of release date) are [31]: 1.1, 1.3, 1.4, 1.5, 1.4.2, 2.0, 2.1, 2.1.1, 2.1.2, 2.2, 2.3 and 2.4.1 which is the latest version. Among the various versions released, version 2.0 and 2.2 has wider popularity in the industries and has been adopted mostly.

Test Cases

Formally defined, test case is a set of sequential steps to execute a test, operating on a set of predefined inputs to produce certain expected output. Test cases can be manual or automated and can be represented in many forms as document, an assertion statement or set of assertions. Such test cases can be derived from the requirements specified, design model or from the code implemented which aids in testing the software.

In [17], Sommervillie has given the steps in generating the test cases. It is of four steps as: i) *Design Test Cases*, purpose of this step is to generate and prepare a set of test cases where the result is set of test cases, which can be represented in Excel format, as word document or as a database. ii) *Prepare Test Data*, purpose of this step is to generate and prepare test data for each derived test case.

The outcome is a set of test data. iii) *Run program with test data*, an execution test step where the test cases and test data will be run which results in actual system output. iv) *Compare results to test cases*, which compare the system output to expected output in the test cases.

There are various techniques to generate test cases are: random techniques, goal-oriented techniques that covers selected goal such as a branch or statement, specification-based techniques, sketch diagram based techniques that derives test cases from UML diagrams, source code-based techniques also called as path-oriented technique that identifies the path to be covered for generating the test cases. This paper deals with the assorted test case generation techniques from UML design models.

UML Diagram 2.2 Overview

The UML diagrams are categorized based on two hierarchy levels. i) Structure Diagrams show the static structure of the system and how they are related to each other. ii) Behavior Diagram provides an explanation of the system as a whole. They model the behavior of system in the form of diagrams, which can be described as a series of changes to the system over time. The categories of both kinds are depicted in the Fig. 1.

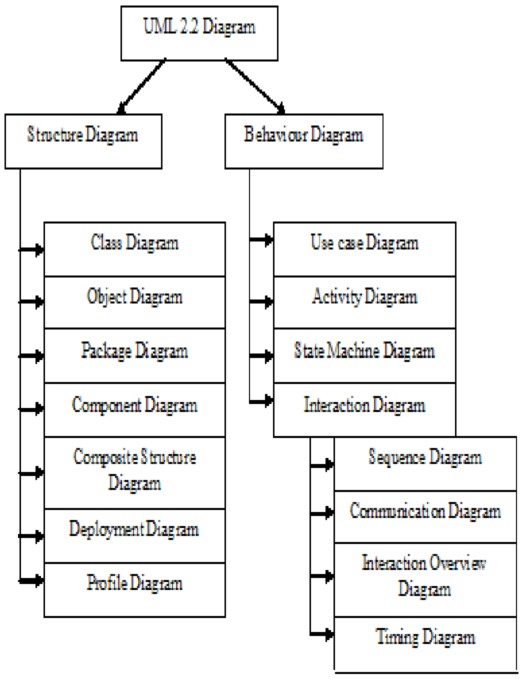

Fig. 1. UML diagrams

III. LITERATURE SURVEY

The UML model technique is most widely used in software design phase. The literature review shows that UML diagrams are typically used to automatic generate test cases. A number of approaches for generating test cases in a roundabout way from the UML analysis and design models, i.e. use case diagram, sequence diagram, collaboration diagram, class diagram, state chart diagram or a combinational approach are proposed. But most of them need to translate UML description into another description such as a graph (model based testing) or a table and then derive the test cases. This paper brings few such existing techniques.

In [1] Jeevarathinam and Antony Selvadoss Thanamani have proposed automation generation of test cases from software specification. Test inputs can be derived by two major approaches as static approach, in which inputs can be generated from models of the system, where as dynamic approach generates tests by repeatedly executing the program.

They presented a framework for testing based on automated test case generation using input specification.

The frameworks combine symbolic execution and model checking techniques for the verification of java program. Test coverage criteria used is branch coverage criteria. The framework of their approach is given in Fig.2.

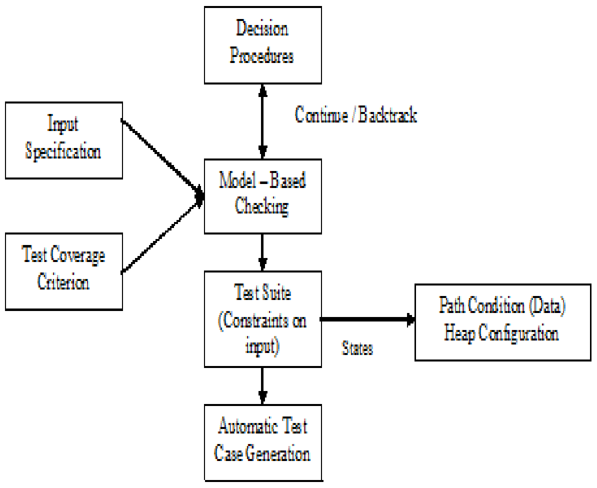

Fig. 2. Automated test case generation [1]

According to the input specification, the model is represented as symbolic execution tree and test coverage criteria which is given as non-deterministic program, represents each path condition. The model checker explores the program symbolic state space placed with constraints on inputs (heap configuration and path condition). The decision procedure guides in checking for the correct path to be covered. Whenever a violation occurs, model based checking can backtrack and could take another available path to generate the automatic test cases. The major limitation of the presented framework works only for java byte code, and is not generic.

In [2], Prasanna et al. have made a survey on automated test case generation techniques categorized under specification based test case generation and model base test case generation. They also touched on few other approaches for generating test cases as path oriented test case generation and intelligent techniques.

Test Case Generation from UML Structured Diagram

In [3], Prasanna and Chandran proposed a new model based approach for automated test cases for UML object diagrams using genetic algorithm.

In their work, Object diagram of a banking system created using Rational Rose tool has been considered for test case automation process. The methodology followed was a tree based approach in which initially the object diagram was constructed using rational rose software and it was parsed using java swing to capture the object names. A tree was built where objects are represented as nodes, attributes (object inputs) forms the left branch of the corresponding node and methods (object outputs) forms the right branch of the corresponding nodes. In case of duplication, it is added as a prefix for the nodes to form a general tree. Later this tree is converted into binary tree and depth first search technique was applied on it to generate the test case set which gave the valid, invalid and termination sequences for the application. In order to prove the effectiveness of their approach, mutation analysis was carried on where faults were injected deliberately to reveal the presence of faults. Their experimental results have shown that it has the capability to reveal 80% fault in the unit level and 88% fault in the integration level.

In [4], Shanthi and Mohan Kumar proposed a technique for generating test cases for object oriented software that uses data mining concepts to design and derive test cases. The data mining technique was applied to generate optimal test cases. The proposed approach used UML Class diagram, in which a tool is used to extract the information from the diagram and are mapped into a tree structure. A cross over operator was applied to discover all patterns from the tree. They followed an approach similar to [3].

Test Case Generation from UML Behavioral Diagram

Noraida Ismail [18] et al. proposed automatic test case generation from UML use-case diagram intended to reduce the cost of testing the system. A simple use-case diagram for online bookstore system where only a registered user (customer) is allowed to place an order for available items on the web is considered in their work. They built a GenTCase tool that is able to generate the test cases automatically according to the system's requirements. At first, the requirements are transformed into a UML use-case diagram, and then the test cases were automatically generated according to the use cases.

The test cases can be used as a checklist for a programmer to validate the system whether it meets the requirements. The major limitation of the built tool GenTCase is that it captures only the functional requirements of the system where as it fails to capture the non-functional requirements of the system.

Puneet Patel and Nitin N. Patel [5] proposed an approach to generate test case using UML Activity diagram and also a coverage criterion called activity path coverage criterion. A graph based approach was followed to generate the test cases. The activity diagram for registration cancellation use case was converted into a directed activity graph. In the graph, the constructs in the activity diagram is represented as node and the flow in the activity diagram is represented as edge. Test cases from activity graph are generated using activity path coverage criterion, by traversing the paths in breadth first search. An activity path is a path in an activity graph that considers a loop at most two times and maintains precedence relations between concurrent and non concurrent activities. To detect the fault in the loop, general approach is followed as: i) executing loop zero time, ii) executing loop once, iii) executing loop two times, iv) executing loop for n times and v) executing loop for n+1 times for the chosen value of n. Each time the increment or decrement operator of the loop is tested as well as the conditions specified in the loop entry or exit point is tested for proper working.

Pakinam et al. [6] proposed an automated approach for generating test cases from UML Activity diagram. The proposed model created an activity diagram for ATM withdrawal functionality. The activity diagram is used to generate table called Activity Dependency Table (ADT) and converted it into directed graph called an Activity Dependency Graph (ADG). The branch coverage criterion is applied for covering the path and Depth First Search (DFS) traversal technique is applied on the graph for obtaining all the possible test paths. All the details are added to each test path using the ADT to have the final test cases. Fig. 3 shows the architecture of the proposed model that generates test cases from UML Activity Diagram. Each activity diagram should pass through all the four modules: i) Generation of ADT, ii) Generation of ADG, iii) Test cases Generation and iv) On validating the generated test cases at the end, a set of highly efficient test cases meeting the hybrid coverage criterion for the whole system was attained. Validating the generated test cases was achieved by cyclomatic complexity.

International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459 (Online), An ISO 9001:2008 Certified Journal, Volume 3, Special Issue 1, January 2013)
International Conference on Information Systems and Computing (ICISC-2013), INDIA.

The proposed model was applied on three different case studies in three different domains; a University system, a Library system, and an ATM system.

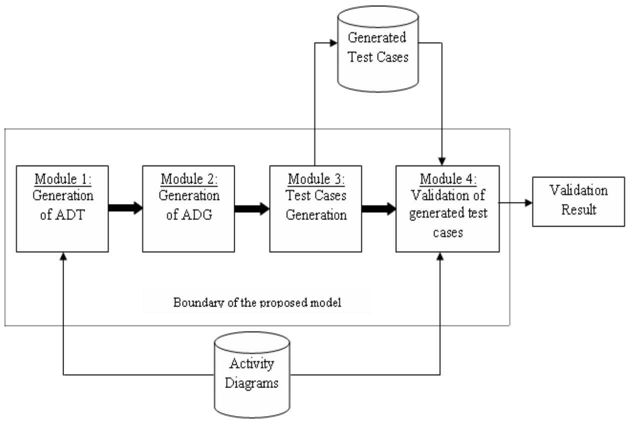
Generated Test Cases
Module 1: Generation of ADT
Module 2: Generation of ADG
Module 3: Test Cases Generation
Module 4: Validation of generated test cases
Validation Result
Boundary of the proposed model
Activity Diagrams

Fig. 3. Generating test cases from UML activity diagram [6]

Debasish Kundu and Debasis Samanta [7] proposed a novel fault based approach to generate test cases from UML Activity Diagram. The activity diagram for registration cancellation use case was considered which is converted into activity graph. An activity graph is a directed graph where each node in the activity graph represents a construct such as initial node, final node, decision node, guard condition, fork node, join node, merge node etc. and each edge of the activity graph represents the flow in the activity diagram. A test coverage criterion called activity path coverage criterion is used to generate test suite. The proposed approach is capable to i) Detect more faults like faults in loop, synchronization faults than the existing approaches. ii) Test case generated might help to identify location of a fault in the implementation, hence reducing the testing efforts.

Luis Fernandez-Sanz and Sanjay Misra [19] proposed a method for automatically generating a complete set of functional test cases for acceptance test level taking software specifications with use cases and complementary activity diagram as a starting point. Also a tool is developed as plug-in for the Eclipse open environment that allows easy application in real practice. Andreas Heinecke et al. [20] have proposed a technique using UML Activity diagram involving the case study of traveller problem and generate test cases from specification.

This approach used black box testing type and used applied modified depth first search algorithm. All path coverage criterions are used.

Zhang Mei et al. [8] proposed an approach to generate automated test case from UML Activity diagrams using ATM System. A tool called TSGen was used. The proposed model generated the test case from test scenario using concurrency coverage criteria. In [21], Hyungchoul Kim et al. have proposed a graph based model that is based on an I/O explicit Activity Diagram (IOAD) to generate test cases from UML activity diagrams with an aim to minimize the number of test cases. IOAD model is an abstraction model obtained from the fully expanded activity diagram by exposing only external inputs and outputs. The technique followed to derive test case from UML activity diagrams were: i) Derive an activity diagram from the given specification. ii) Convert the activity diagram into a directed graph. iii) Test scenario and test cases were extracted from the graph using all-paths test coverage criterion.

Suppandeep Sandhu and Amardeep Singh [9] proposed a technique for generating test case based on gray box testing with UML Activity Diagrams, traversed in Depth First Search manner. This approach is used to generate test cases directly from UML activity diagram using java programming, and design is reused to avoid the cost of test model creation. Test cases were generated using PETA java/eclipse based platform an automated software testing tool. The tool is highly productive and flexible. The activity diagram was designed for tax calculator whose inputs were employee code, medical claim amount, investment claim amount and salary. The output was the tax paid by the employee. The procedure followed for the generation of test case is three fold: i) Firstly apply settings under the tools. ii) Class diagrams are generated. iii) From class diagram, activity diagram is created and finally test cases are generated. A method that combines the white box method and black box method is called as gray box method. It extends the logical coverage criteria of white box method and finds all the possible paths from the design model, which describes the expected behavior of an operation. Then it generates test cases, which can satisfy the path conditions by black box method. The basic path is defined based on coverage criteria of gray box method as the test Completion criteria.

In order to get all basic paths, the activity diagram is traversed by DFS (Depth First Search) method from the initial activity to the final activity.

In [10], Mingsong Chen et al. have proposed an approach to derive test cases from a set of randomly generated ones according to a given test adequacy criterion, instead of directly deriving from the UML Activity diagram. A case study of Online Stock Exchange System (OSES) was considered as an example in their work. In the approach, they first instrument a Java program under testing according to its activity diagram model, and randomly generated abundant test cases for the program. Then, the instrumented code is executed to obtain the corresponding program execution traces. Finally, these traces were matched with the activity diagram to generate a reduced set of test cases according to the given test adequacy criterion. An interactive graphical tool is also developed which allows the user to construct, edit and analyse activity diagrams.

In [22], Johannes Ryser and Martin Glinz proposed an approach named the SCENT-Method (A Method for SCENario-based Validation and Test) of software, to create scenario in the analysis phase and the same can be used to test the system by determining the test cases. Test cases for system test were generated by path traversal in the statecharts. The proposed approach used scenarios not only to elicit and document requirements, they were also used to describe the functionality and specify the behaviour of the system, also validated the system under development while it is being developed.

Chartchai Doungsa-ard et al. [23] have proposed a technique to generate test data from UML state diagram using genetic algorithm which helps in generating test cases before coding. A coffee vending machine example is used in their approach. In [24], Kansomkeat and Rivepiboon, introduced a method for generating test sequences using UML state chart diagrams. They transformed the state chart diagram into a flattened structure of states called testing flow graph (TFG). From the TFG, they listed the possible event sequences which they considered as test sequences. The testing criterion used to guide the generation of test sequences is the coverage of the states and transitions of TFG.

In [16], Prasanna et al. have proposed a new model-based automatic test case generation for object-oriented systems using UML Collaboration diagram (Communication diagram in UML 2.2) coupled with mutation analysis to describe the effectiveness of their work. They used buying tickets as case study, which gets the cash from the customer and returns back the balance and the ticket to them. The whole function of the system was represented in the form of collaboration diagram. Their methodology involved construction of collaboration diagram using rational rose, converting it into a weighted graph where objects in the diagram falls as nodes and messages as edges. The sequence number denotes the weights between two appropriate nodes. In order to bring all possible test cases from the graph Prim's algorithm and Dijkstra's algorithm were used to find the adjacency node with minimum-weighted edges. All valid, invalid and termination sequences of the application were obtained until maximum-weighted edge was reached.

In [25], Samuel et al. proposed an automatic test sequence generation method that derived test sequence from state machine diagrams. The state machine diagrams are one of the extended UML state chart diagram. Their methodology has three main steps in the generation, which are: i) to select a predicate on a transition from the state diagram, ii) to transform the function and iii) generate a test sequence based on the transformed function.

In [26] Shanthi and Mohan Kumar have proposed test cases generation by using UML Sequence diagram using genetic algorithm. The best test cases were optimized and validated by prioritization. The proposed method is evaluated by the Sequence diagram of Banking System created using rational rose is used for generating automated test case from it. The methodology followed was: i) Created a sequence diagram using Rational Rose saved in a uml model file with .mdl as extension. ii) Extracted the necessary information from the file by writing a parser program in java. iii) Sequence Dependency Table (SDT) was generated based on the extracted information. iv) Test path were generated and by applying genetic algorithm test cases were prioritized. Fig. 4 shows the flow diagram for test case generation from UML Sequence diagram using genetic algorithm.

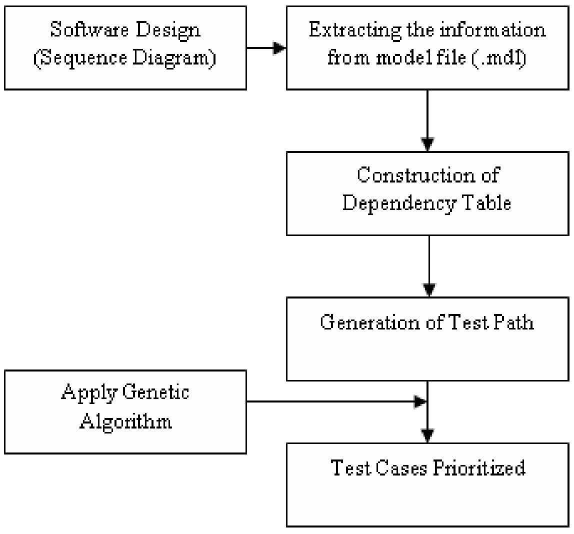
Software Design (Sequence Diagram)
Extracting the information from model file (.mdl)
Construction of Dependency Table
Generation of Test Path
Apply Genetic Algorithm
Test Cases Prioritized

Fig. 4. Flow diagram for test case generation from UML sequence diagram using genetic algorithm [26]

In [27] Emanuela et al. have proposed a model based testing technique to generate test cases from UML sequence diagrams that are translated into Labeled Transition Systems (LTSs) early in the development phase in order to reduce the cost involved. An example of Motorola mobile phone application was selected. The procedure followed is: i) First LTS model for functional testing was obtained from sequence diagram and ii) Test cases were derived from the obtained model. The LTS was traversed from its initial state to end state using Depth First Search (DFS) method to obtain the test path.

Test Case Generation by a Combinational Approach

Saru Dhir [11] analyzed the UML Structural and Behavior diagrams. He proposed UML Class diagram and object diagram (structural model) can be used to generate minimal test cases. The technique used is first the class diagram and object diagrams were modeled, then coded, and the test cases are generated from both based on the model information. Different test cases are generated by object diagram and class diagram. A comparison between both these diagrams is made in order to identify the redundant test cases. Then with the help of instrumented code, the information as how many numbers of times the code is tracked is found.

Using this information, the number of test cases were reduced which resulted in the minimal test cases, on removing the redundant ones.

Monalisha Khandai et al. [12] have given a survey on test case generation from UML models. They proposed a framework for generating test cases from combinational UML models. Different UML models are considered, that are converted into an intermediate formats. The test cases are generated from the combinational intermediate format. Their technique is capable of detecting more number of faults than compared to single UML models. Fig. 5 shows the framework of their approach for generating test cases from combinational UML models.

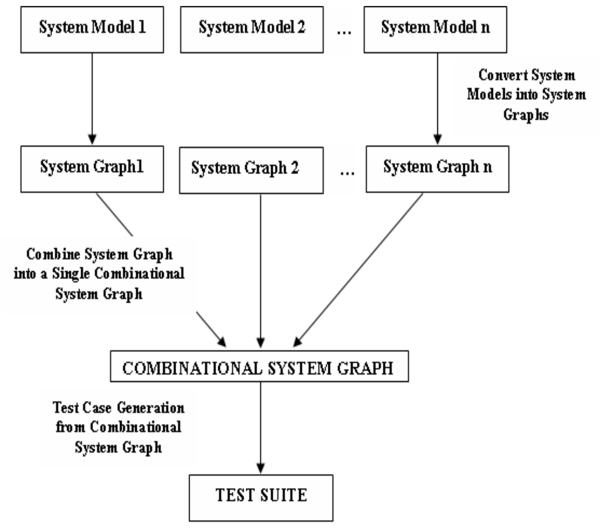
System Model 1
System Model 2
...
System Model n
Convert System Models into System Graphs
System Graph1
System Graph 2
...
System Graph n
Combine System Graph into a Single Combinational System Graph
COMBINATIONAL SYSTEM GRAPH
Test Case Generation from Combinational System Graph
TEST SUITE

Fig. 4. Framework for generating test cases from combinational UML models [12]

Vinaya Sawant and Ketan Shah [13] proposed graph based methodology to generate test cases from various UML diagrams as use case diagram, class diagram, use case templates, sequence diagram and data dictionary using OCL (Object Constraint Language), which are associated with the use case for which the sequence diagram is considered. A case study of Bank ATM system (Withdraw money) is considered. The methodology used was they converted the UML sequence diagram into a graph called Sequence Diagram Graph (SDG), which is traversed in breadth-first search manner to enumerate all the paths to generate test cases.

The information that is required for the specification of a test case such as input, pre condition, post condition and output are retrieved from the extended use cases. The data dictionaries expressed in class diagrams are stored in the SDG. They also developed a tool ATCUM (Automatic Test Case generation from UML Models) for generating test cases. The implementation was done in Java programming language by following the steps as: i) Drawing UML Diagrams from the problem statement, ii) Generating XML File, iii) Parsing XML file, iv) Generating Scenarios, v) Display graph, vi) Test Set Generation and vii) Creating and saving temporary file for the software tester usage. One major drawback of the proposed methodology is the XML file generated needs to be edited according to the requirements, in case if the requirements changes. Fig. 6 shows the schematic block diagram for test case generation.

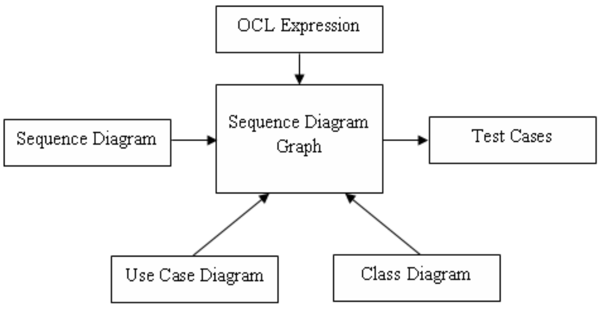

Fig. 6. Combinational approach for test case generation [13]

Arpita Tewari and Misra [14] proposed a novel approach based on the combination of UML diagrams activity diagram, sequence diagram and communication diagram for generating the test cases. A graph has been constructed and depth first search graph traversal technique has been used to generate test cases. The approach used was, the activity diagram, sequence diagram and communication diagram for ordering system designed in Rational Rose Software were converted into separate graphs. Later all these graphs were integrated into a single graph by using the proposed algorithm. The graph was traversed in depth first manner to generate test cases. The major limitation of the proposed approach was no valid test cases were able to generate apart from the test cases intended to generate. The system was not able to solve integration operation faults that arose while integrating the designed three graphs into a single graph. Fig. 7 shows the combinational approach to generate test cases.

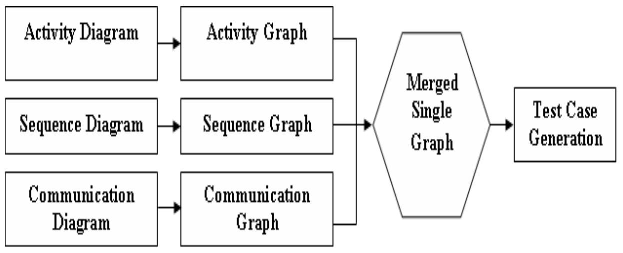

Fig. 7. Combinational approach for test case generation from UML activity diagram, sequence diagram and communication diagram [14]

Supaporn Kansomkeat et al. [28] presented a comparative evaluation of tests generated from UML statecharts and UML sequence diagram. Their work presented data comparing the use of statecharts and sequenced diagram for unit testing and integration testing. Statechart diagram describes software behaviors with states and state transitions of individual software components. They are naturally considered to be a good source for unit testing. Sequence diagrams describe interactions among software components, and thus are naturally considered to be a good source for integration testing. The framework suggested achieved its goal by generating test cases from UML statechart and UML sequence diagrams were listed both at unit level and integration level testing. Their fault reviling capabilities were compared. As an experimental approach they obtained three results: i) Statechart test sets are better at discovering unit level faults than sequence diagram test sets. ii) Sequence diagram test sets is better at discovering integration level faults than statechart test sets. iii) Statechart results in more test cases than sequence diagram. They modelled the software for a typical cell phone hand set system that includes class diagrams of six classes, five statechart diagrams and six sequence diagrams. Implementation was carried using java programming language. 81 test cases were generated from statechart and 43 from sequence diagram derived manually. Limitation of this approach is due to concurrence problem, the generations of test cases were not able to be automated.

In [29], Dehla Sokenou focused on a technique for generating test cases from a combination of UML sequence and state diagrams. A bank account is used as example in their work. Test cases are generated for class and integration testing.

The main information is extracted from sequence diagrams and then it is complemented by state diagram. Each sequence diagram defines a set of test cases differing in terms of the states of the participating objects. Their technique can be applied to positive test cases, negative test cases and test system can generate complementary test cases. Complementary test cases consider states in which a sequence violates the implicit precondition. However they recommend the usage of complementary test cases as an optional one.

In [30], Yiwen Wang and Mao Zheng has proposed a methodology to generate the test case from a design level class diagram and an interaction diagram, especially for a large, complex system. In large complex system modeling, the class diagram can be used to provide the static configurations of the system whereas the interaction diagram can be used to describe the dynamic behavior of the system. A car rental example is used to illustrate the test case generation. The class diagram defines the object configuration and the interaction diagram determines the method sequence in the testing. The generated test cases are compared with a few other UML model-based test case generation methodologies; they are able to meet the required test adequacy criteria better.

In [15], Santosh Kumar Swain et al. have proposed a test case generation by combining static and dynamic behavior of the UML model. They used use case and sequence diagrams of UML 2.0 to derive test cases for integration level testing object-oriented software. Using UML sequence diagrams and use case diagram, Use case Dependency Graph (UDG) was constructed from use case diagram and Concurrency Control Flow Graph (CCFG) was constructed from sequence diagrams for test sequence generation. The graphs are traversed with an aim to cover the various interaction faults, message sequence faults and synchronization faults. The approach used XML and a semi-automated tool called ComTest has also been developed that works based on coverage criteria. The Fig. 8 shows the overview of ComTest tool.

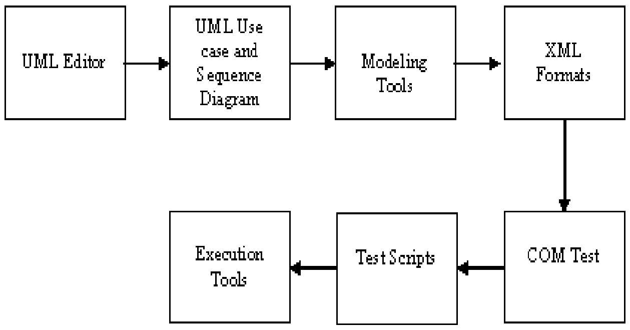

Fig. 8. Overview of ComTest tool [15]

IV. OVERVIEW OF UML MODELING TOOLS

The most well-known UML modeling tool is IBM Rational Rose. Factors that should be considered when choosing UML tools are: i) Have a clear understanding of what features are important to create models. ii) Select tools in such a was as it support most UML analysis diagrams such as class diagram, use-case diagram, collaboration diagram, sequence diagram and activity diagram. iii) Narrow down the tools and try for a demonstration tool. iv) The tool should be easy to use, reliable and scalable.

Enormous amount of UML tools are widely available. Some open source UML modeling tools are: Umbrello, Astade, FUJABA, AgroUML, and Coral. Various open source drawing tools are: DIA, Violet, UMLet. The various commercially available modelling tools that support UML Diagrams are: Magic Draw, IBM Rational Rose, JUDE, gModeler, Rhapsody, Modelistic, Visual thought, EclipseUML, UMLStudio, SmartDraw, MetaEdit+, Select Component Architect, Visual Paradigm for the Unified Modeling Language (VP-UML), SequenceSketcher, EctoSet Modeller, ProxyDesigner, JVision, iUML, Embarcadero Describe, WinA&D, MacA&D, HAT (HOORA Analysis Tool), ObjectDomain, Visual UML and Together. Few drawing tools are: OmniGraffle and Visio.

V. CONCLUSION

Unified Modeling Language (UML) has now become a de facto model in the field of software testing, particularly in the industry sectors. New technique for the generation of test case from these UML diagrams needs to be identified. Also, the study shows that existing methods mostly concentrates on the behavioral diagrams, in which some does not work for maximum test coverage where as some methods prepare and generate a significant number of tests with less test coverage. Much of the existing techniques very few concentrate on redundant test path generation and no solution have been proposed for eliminating redundant test cases. In future, we have planned to develop a test path method that minimizes size of tests, time and cost, while preserving maximum test coverage using Modified Condition / Decision Coverage criterion.

REFERENCES

1. Journal Papers

[1] Jeevarathinam, Antony Selvadoss Thanamani. 2010. Towards Test Case Generation from Software Specification. International Journal of Engineering Science and Technology, Vol. 2(11), pp. 6578-6584.

[2] Prasanna M., S. N. Sivanandam, Venkatesan, R. Sun-darrajan. 2005. A Survey on Automatic Test Case Generation. Academic Open Internet Journal.

[3] Prasanna M. and K.R. Chandran. 2009. Automatic Test Case Generation for UML Object diagrams using Genetic Algorithm. International Journal on Advance Soft Computing Application, July, Vol. 1, No. 1.

[4] Shanthi A.V.K. and G. Mohan Kumar. 2011. Test Cases Generation for Object Oriented Software. Indian Journal of Computer Science and Engineering (IJCSE), Aug-Sep, Vol. 2, No. 4.

[5] Puneet Patel and Nitin N. Patel. 2012. Test Case For-mation using UML Activity Diagram. World Journal of Science and Technology, pp. 57-62.

[6] Pakinam N. Boghdady, Nagwa L. Badr, Mohamed Hashem and Mohamed F.Tolba. 2011. A Proposed Test Case Generation Technique Based on Activity Diagrams. International Journal of Engineering and Technology, June, Vol. 11, No. 03.

[7] Debasish Kundu and Debasis Samanta. 2009. A Novel Approach to Generate Test Cases from UML Activity Diagrams. Journal of Object Technology, June, Vol. 8, No. 3, pp. 65-83.

[8] Zhang Mei, Liu Chao, Sun Chang-ai. 2010. Auto- mated Test Case Generation Based on UML Activity Diagram Model. Journal of Beijing University of Aeronautics and Astronautics (in Chinese), Vol. 27, No. 4, pp. 433-437.

[9] Suppandeep Sandhu, Amardeep Singh. 2011. A Systematic Approach for Software Test Cases Generation using Gray Box Testing with UML Activity Diagrams. International Journal of Computer Science and Technology, Oct-Dec, Vol. 2, Issue 4.

[10] Mingsong Chen, Xiaokang Qiu, Wei Xu, Linzhang Wang, Jianhua Zhao And Xuandong Li. 2007. UML Activity Diagram-Based Automatic Test Case Generation for Java Programs. The Computer Journal Advance Access, August 25.

[11] Saru Dhir, 2012. Impact of UML Techniques in Test Case Generation. International Journal of Engineering Science and Advanced Technology, March–April. Vol. 2, Issue 2, pp. 214-217.

[12] Monalisha Khandai, Arup Abhinna Acharya, Durga Prasad Mohapatra, 2011. A Survey on Test Case Generation from UML Model. International Journal of Computer Science and Informational Technologies, Vol. 2(3).

[13] Vinaya Sawant, Ketan Shah, 2011. Automatic Gen-eration of Test Cases from UML Models. International Conference on Technology Systems and Management (ICTSM). Proceedings published by International Journal of Computer Applications (IJCA).

[14] Arpita Tewari and A. K. Misra. 2011. A Novel Approach to Generate Test Cases using UML Diagrams. International Journal of Software Engineering, Vol. 2, No. 2, pp. 109–124.

[15] Santosh Kumar Swain, Durga Prasad Mohapatra, and Rajib Mall. 2010. Test Case Generation Based on Use case and Sequence Diagram. International Journal of Software Engineering, July, IJSE, Vol.3, No.2.

[16] Prasanna M., K. R. Chandran., K. Thiruvenkadam. 2011. Automatic Test Case Generation for UML Collaboration. IETE Journal of Research, Vol. 57, Issue 1, pp.77-81.

2. Text Book

[17] Sommervillie, I., 2000. Software Engineering, 6th Edition, Addison-Wesley, England.

3. Conference Proceedings

[18] Noraida Ismail, Rosziati Ibrahim, Noraini Ibrahim, 2007. Automatic Generation of Test Cases from Use-Case Diagram. Proceedings of the International Conference on Electrical Engineering and Informatics Institute Technology, Bandung, Indonesia, June 17-19.

[19] Luis Fernandez-Sanz, Sanjay Misra, 2012. Practical Application of UML Activity Diagram for the Generation of Test Cases. Proceedings of the Romanian Academy, Series A, Vol. 13, No. 3, pp. 251-260.

[20] Andreas Heinecke, Tobias Bruckmann, Tobias Griebe, Volker Gruhn, 2010. Generating Test Plans for Acceptance Tests from UML Activity Diagrams. 17th IEEE International Conference and Workshops on Engineering of Computer-Based Systems, Vol. 14, pp. 425-654.

[21] 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.

[22] Johannes Ryser, Martin Glinz, 1999. A Scenario-Based Approach to Validating and Testing Software Systems Using Statecharts. Presented at the 12th International Conference on Software and Systems Engineering and their Applications ICSSEA'99. Proceedings: CNAM, Paris, France.

[23] Chartchai Doungsa-ard, Keshav Dahal, Alamgir Hossain, and Taratip Suwannasart, 2007. An Automatic Test Data Generation from UML State Diagram using Genetic Algorithm. The proceedings of the Second International Conference on Software Engineering Advances.

[24] Kansomkeat, S. and W. Rivepiboon, 2003. Auto-mated - Generating Test Case using UML Statechart Diagrams. In Pro-ceeding SAICSIT 2003, ACM, pp. 296 – 300.

[25] Samuel P., R. Mall and A. K. Bothra, 2008. Auto-matic Test Case Generation Using Unified Modeling Language (UML) State Diagrams. IET Software, Vol.2, Issue 2, pp. 79-93.

[26] Shanthi A.V.K. and G. Mohan Kumar, 2012. Auto-mated Test Cases Generation from UML Sequence Diagram. International Conference on Software and Computer Applications (ICSCA 2012), Vol. 41, Singapore.

[27] Emanuela G. Cartaxo, Francisco G. O. Neto and Patrıcia D. L. Machado, 2007. Test Case Generation by means of UML Se-quence Diagrams and Labeled Transition Systems. Proceedings of the IEEE International Conference on Systems, 7-10, October, Man and Cybernetics, Montréal, Canada.

[28] Supaporn Kansomkeat, Jeff Offutt, Aynur Abdurazek, Andrea Baldin, 2008. A Comparative Evaluation of Tests Generated from Different UML Diagrams. Ninth ACIS International Conference on Software Engineering.

[29] Dehla Sokenou, 2006. Generating Test Sequences from UML Sequence Diagrams and State Diagrams. Proceedings of GI Jahrestagung (2), pp. 236-240.

[30] Yiwen Wang and Mao Zheng, 2012. Test Case Generation from UML Models. 45th Annual Midwest Instruction and Compu-ting Symposium, Cedar Falls, Iowa.

4. Generic Website

[31] www.uml-diagrams.org