AMSE JOURNALS –2015-Series: Advances D; Vol. 20; N° 1; pp 1-20

Submitted Feb. 2015; Revised June 2, 2015; Accepted July 25, 2015

A Systematic Survey on Automated Test Case Generation from UML Sequence Diagram Using GA and Other Approaches

* T. Tamilarasi, ** M. Prasanna.

*School Of Information Technology and Engineering, Research Associate,
VIT University, India, Vellore-632014, (tamilarasi.tarasu@vit.ac.in)
**School Of Information Technology And Engineering, Associate Professor,
VIT University, India, Vellore-632014. (prasanna.m@vit.ac.in)

Abstract

Software testing is the pre-eminent part of the software development life cycle process. Generating test cases is one of the challenging parts in software testing. Test cases are a set of conditions where a tester will test whether the system or an application is working as it is established to do. Test cases are often referred as *test scripts*, when they are collected into *test suite*. Test automation defines that the system should work based on the established functionalities that has been described. Genetic Algorithm is an adaptive search optimization technique which performs implicitly parallel search in a large solution space and manipulates simulation. The nature of GA is to perform testing of all competitive behavior of the solution space until a good behavior is evolved. This paper presents a survey on automated test case generation from UML sequence diagrams using GA (Genetic Algorithm) and other techniques.

Keywords: GA (Genetic Algorithm), Software Testing, Test Case, UML Sequence Diagram, Test Scripts, Test Suite.

1. Introduction

Software testing [27] is the process of evaluating a system or its components with the intent to find that whether it satisfies the specified requirements or not. Testing is an activity associated with any process that produces a product. Testing consumes 50% of the development budget. Testing is executing a system in order to identify any gaps, errors or missing

requirements in contrary to the actual desire or requirements. William E. Perry [26] describes more about Software Testing. The course of software being tested in a well-planned way is known as Software testing life cycle. Figure-1 explains the life cycle of software testing. A test case is a set of conditions or variables under which a tester will determine whether a system under test satisfies requirements or work correctly. The process of developing test cases can also help find problems in the requirements or design of an application. A test case has components that describe an input, action or event and an expected response, to determine if a feature of an application is working correctly. A test case is basically a description of test. A good test case should have some characteristics which are: (i) it is reusable, (ii) should be accurate and tests what it is intended to test and (iii) it should be traceable to requirements.

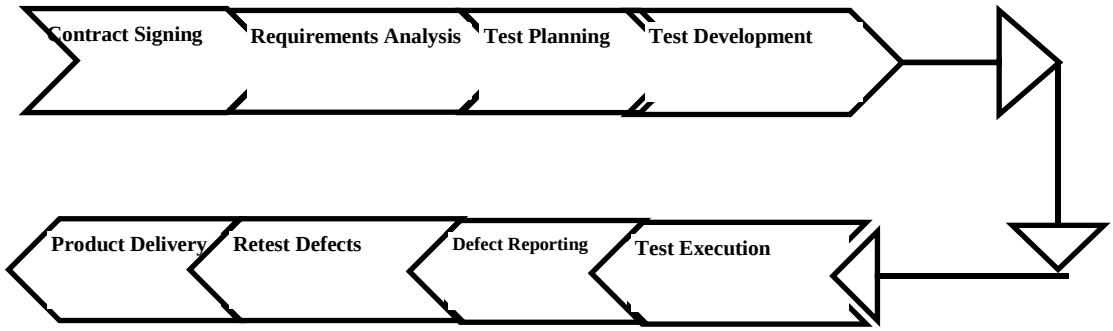

Figure 1: Software Testing Life Cycle Process

Jim_Arlow et al [8] describes more about UML: The Unified Approach. The Unified Modeling Language (UML) is a standard language for writing the blueprints of software. The UML may be used to visualize, specify, construct and document the artifacts of a software system. Here, we discuss about the behavioral aspects of the UML diagrams. Behavioral aspects are the dynamic parts of UML models, which represent behavior over time and space. UML Sequence diagram is an interaction diagram that emphasizes the time-ordering of messages. A Sequence diagram shows an interaction, consisting of a set of objects and their relationships, including the messages that may be dispatched among them. Graphically, a sequence diagram is a diagram that shows objects arranged in X axis and messages arranged along Y axis. Unlike sequence diagram, no other UML diagram shows the lifeline of objects explicitly. GA [29] is a search heuristic that traverses the process of natural selection. The meta-heuristic method is routinely used to generate useful solutions to optimization and search problems. On analyzing several approaches, GA technique serves as efficient enough as other techniques that are involved in automated test case generation.

2. Background

In object-oriented software [28], the complexity of the software is not only associated with functions and procedures but also with how the procedures and classes are connected and how objects communicate. There are two kinds of aspects in UML: Structural aspects and Behavioral aspects. Structural aspects are the nouns of UML models. These are the static parts of a model, representing elements that are either conceptual or physical. Behavioral aspects are the dynamic parts of UML models. These are verbs of a model, representing behavior over time and space. A sequence diagram is basically a projection of the elements found in a message flow. The semantics of an interaction's context, objects and roles, links, messages and sequencing apply to sequence diagrams. A sequence diagram emphasizes the time ordering of messages. There are several approaches have been proposed for test case generation from UML sequence diagrams. In this paper, survey has been done on the work carried out on behavioral aspects of UML sequence diagram using GA and other approaches.

3. Genetic Algorithm: A Detailed Description

Genetic algorithm is one such algorithm in evolutionary algorithm. GA is a search algorithm that is inspired by the way nature evolves species using natural selection of the fittest individuals. Franz Rothlauf [5] explains more about evolutionary algorithms like genetic algorithms. The possible solution to a problem is represented by the population of the chromosomes. A chromosome is a string of binary digits and each digit that makes up a chromosome is called a gene. The initial population can be totally random or can be created manually using processes such as greedy algorithm. GA uses three operators in its population namely: (i) Selection, (ii) Crossover and (iii) Mutation.

3.1 Selection

A selection scheme is applied to determine how individuals are chosen for mating based on their fitness. Fitness can be defined as a capability of an individual to survive and reproduce in an environment. Selection generates the new population from the old one, thus starting a new generation. Each chromosome is evaluated in present generation to determine its fitness value. The fitness value is used to select the better chromosomes from the population for the next generation.

3.2 Crossover

After selection, the crossover operation is applied to the selected chromosomes. It involves swapping of genes or sequence of bits in the string between two individuals. The process is repeated with different parent individuals until the next generation has enough individuals. After crossover, the mutation operator is applied to a randomly selected subset of the population.

3.3 Mutation

Mutation alters chromosomes in small ways to introduce new good traits. It is applied to bring diversity in the population.

4. Generating Test Case Using Genetic Algorithm

A.V.K. Shanthi et al [1] describe automated test case generation by means of UML sequence diagram using Genetic algorithm with best test cases optimized and validated by prioritization. Their approach is significant to identify location of a fault in the implementation, thus reducing testing effort. Moreover their method for test case generation inspires the developers to improve the design quality, find faults in the implementation early, and reduce software development time. Figure-2 briefly explains the methodology of test case generation.

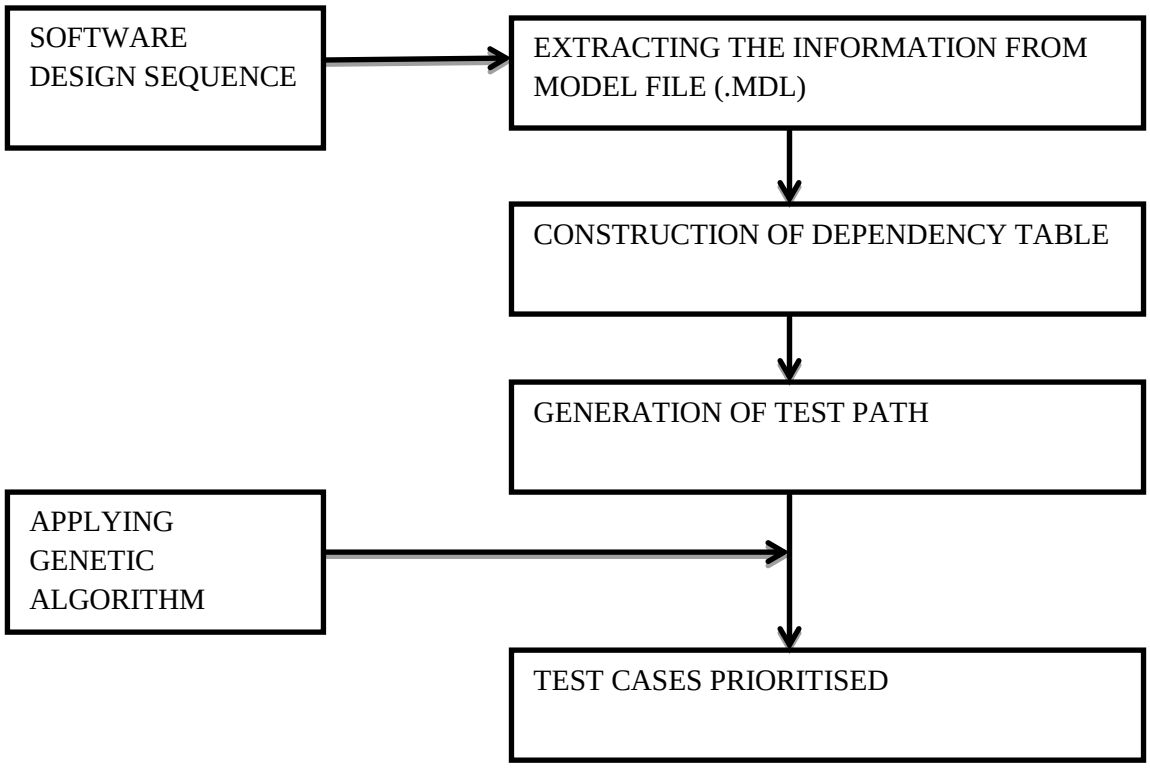

Figure 2: A.V.K. Shanthi et al [1] Methodology for test case generation

Paramjit Kaur et al [15] describes about the various techniques to generate the test cases from the UML diagrams to test the software automatically. A new method is proposed in generating test cases by testing the software automatically using Genetic Algorithm Crossover Technique. Their technique will be applied on sequence diagram. Their method is coupled with mutation testing to check the effectiveness of the methodology. The results show that when Genetic Algorithm combined with mutation testing reveals 80% of the effectiveness in testing process. So that, the errors are validated in the design phase itself early in the development process stage.

V. Mary Sumalatha et al [24] describes about the test case generation for object oriented software using UML diagrams like Sequence diagram. The proposed methodology of their paper is as follows: (i) Converting sequence diagram into Sequence Diagram Graph, (ii) Applying Genetic algorithm to sequence diagram graph, (iii) generate all paths between source and destination as loops, (iv) Calculate fitness value by calculating the probability of the individual, (v) To calculate the probability of the individual, select the individuals from the large initial population, (vi) produce new generation of solutions by generating random values , (vii) Perform crossover technique. (viii) Mutate if random value is less than 0.2 to obtain best test path, (ix) re-evaluate the fitness for new generation, (x) repeat the process until all paths between source and destination have been covered, (xi) best path is generated from this process. Figure-3 describes steps involved in generating test cases using GA from UML Sequence Diagrams.

Sangeeta Sabharwal et al [18] describe a technique to prioritize test case scenarios by identifying the critical path clusters using genetic algorithm. Their proposed technique has extended our previous work of generating test case scenarios from activity diagram by also considering the concurrent activities in nested activity diagram. Their technique involves three algorithms. Algorithm1 describes about test data generation by GA from activity diagram. Algorithm 2 is a procedure call for Algorithm1, where that algorithm traverses CFG which is generated from activity diagram. For each node in CFG, some weight has to be assigned. Algorithm2 is defined for assigning weights for those nodes in CFG. Algorithm 3 describes about test data generation by Genetic Algorithm (GA) from state chart diagram.

M. Prasanna et al [12] describes a new model based approach for automated test case generation in object oriented systems using GA algorithm. This proposed approach involves the following steps: (i) Construct object diagram using rational rose software and store it with .mdl as

extension, (ii) Parse the .mdl file and capture the object names, (iii) Build a tree using object names and apply genetic algorithms cross over technique, (iv) New generation of trees are formed and convert it to binary trees, (v) Traverse new generation of binary trees using Depth First Search technique, (vi) All the valid, invalid and termination sequences of the application can be obtained using Step v. The effectiveness of the generated test conditions can be evaluated using a fault injection technique called Mutation Analysis Technique. It is a fault testing strategy. Programs with injected faults are tested and those faults are called mutants. The outcome of the mutation analysis is a measure named mutation score.

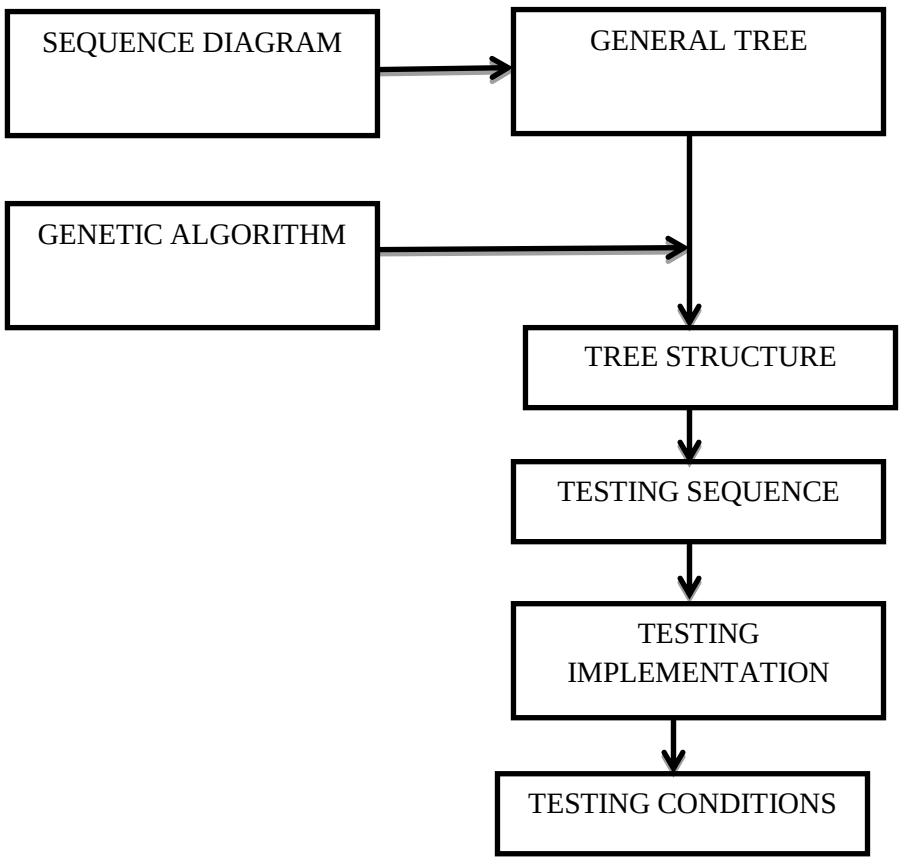

Figure 3: Paramjit Kaur et al [13]. Steps involved in generating test cases using GA from Sequence Diagrams

Score = (Σ faults found / Σ faults injected) * 100---------- (1).

Mutants are obtained by applying mutation operators that introduce the simple changes to original program. The faults are kept in separate versions of the program to avoid interactions

between faults such as masking. The mutation testing has yielded **80.3%** effectiveness in generated test cases in this paper. Figure-4 describes the steps involved in generating test cases.

Hirohide Haga et al [6] proposed a method that automatically generates software test cases based on mutation analysis and GA algorithm. Their proposed methodology include: (i) original test cases are generated randomly under the range of input data, mutants also generated from the code that is to tested, (ii) Evaluate the test cases based on its mutation score whether it is sufficient or not, (iii) the generated test cases are refined by GA algorithm if the mutation score is not sufficient, (iv) Generate High Quality Test Data. Figure-5 describes test case generation based on mutation score.

5. Test Case Generation Approaches

5.1 Random Approach

Random approach is the simplest method for generating test cases. It has the lowest acceptable rate of generating test data. It does not generate quality test cases and does not perform well in coverage criteria. Random approach remains as benchmark among all in test case generation.

5.2 Path wise Approach

It is considered to be one of the best approaches in generating test cases. The approach does not generate by selecting multiple paths but it generate test cases for a specific path that it is intended to work on. By this, the approach reaches a specific knowledge about a specific path and achieves path coverage criteria. From the user, it requires two things: (i) Testing Program and (ii) Testing criteria, i.e. path coverage.

5.3 Goal Oriented Approach

It generates an input for any path from the entry to the exit for a block of code. This reduces the risk of generating test case for infeasible paths. The approach is followed in two methods: (i) Chaining Approach and (ii) Assertion method. Chaining Approach tries to identify a chain of nodes that are vital to the execution of goal node. It starts by execution of the input from the start node. Execution of each branch in the block of code decides the continuation of execution because the current branch does not lead to the goal node. The execution of the input continues in branches until it reaches the goal node. Search algorithms are used automatically to

change input to continue the flow of execution. Hence, it is named as chaining approach. Assertion method is an extension of chaining approach. The major advantage of this approach is that the execution of test case generation can be calculated in the other source rather than coding.

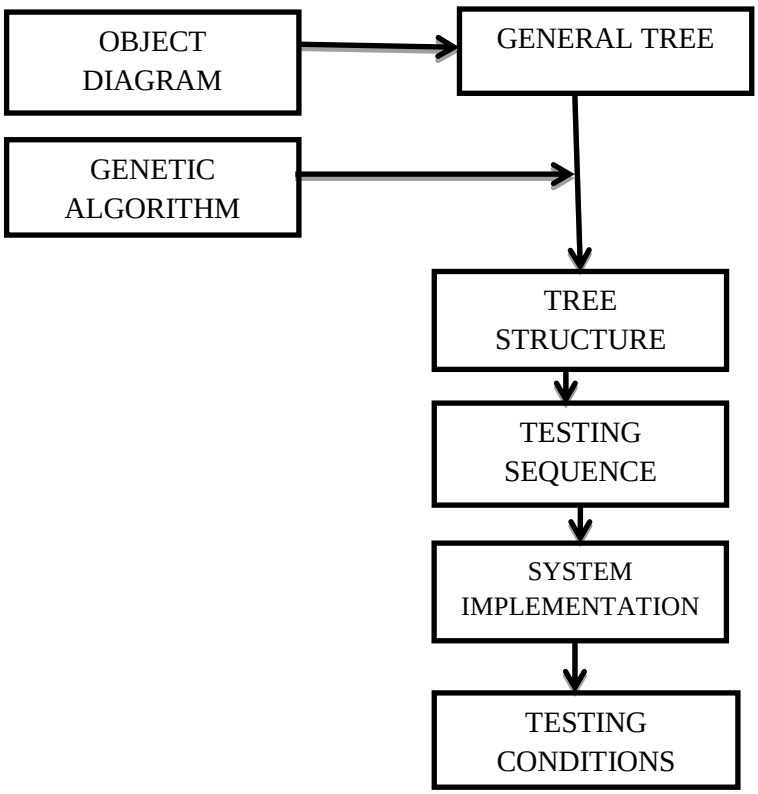

Figure 4: M. Prasanna et al [10] Proposed Methodology for generating test cases from object diagrams using GA algorithm

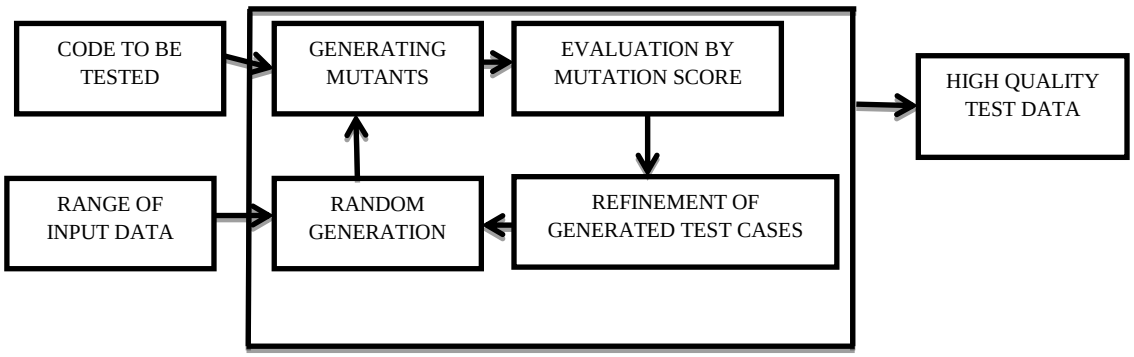

Figure 5: Hirohide Haga et al [5] test case generation using Mutation Analysis and GA Algorithm

5.4 Intelligent Approach

It will generate test cases quicker than other approaches but it analyses a wide range of programs that are quite complex. The test case generation method is coupled with the detailed analysis of the code.

6. Techniques Involved In Automated Test Case Generation

6.1 Generating Test Cases by Condition Slicing

Ranjita Kumari Swain et al [17] describe test case generation using condition slicing and adequate test coverage criteria. Their approach comprises of the these steps: (i) Construction of the UML sequence diagram, (ii) Construction of dependency graph from the sequence diagram, (iii) Selection of conditional predicates from the sequence diagram, (iv) Computation of slices corresponding to each conditional predicate and (v) Generation of test data with respect to slicing criterion. Table-1 explains the test cases generation from slice condition.

Table 6.1: Ranjita Kumari Swain et al [15] Test case Generation from UML Sequence Diagram using Condition Slicing

S.NO	SLICE CONDITION	PREDICATES	TEST CASES
1	(6,s)	s < 100	object1, [11,9], object2.
2	(6,s)	s < 100	object1, [10,10], object2.
3	(7,m)	m <= 120	object1, [120,7], object2.
4	(7,m)	m <= 120	object1, [121,7], object2.
5	(7,m)	m <= 120	object1, [119,7], object2.

The advantage of their approach is, it satisfies high path coverage criteria whereas the disadvantage here is, if the sequence model diagram is large then the path coverage criteria becomes more complex.

6.2 Generating Test Cases from Dependency Graph

Santosh Kumar Swain et al [20] describe constructing use case dependency graph from use case diagram and concurrent control flow graph from sequence diagram for test sequence generation. Their technique can be used for integration and system testing accommodating the object message and condition information associated with use case scenarios. The test cases thus

generated are suitable for detecting synchronization and dependency of use cases and messages, object interaction and operational faults. The drawback here is that testing sequence of messages that are associated with sequence of use cases and look into the details of message sequences within the use case scenario makes integration testing more complex.

6.3 Generating Test Cases with Complete Path Coverage Criteria

Shanmuga Priya et al [23] describe about a model based testing approach from which the test paths are get automated and obtained during the development process. The objective of path coverage criteria is to ensure that whether all the independent paths are executed at least once. A software tester should get an idea that the test cases generated must cover these generated paths in order to ensure complete coverage of the code. Model based testing can test the system under test based on the information of the corresponding model. It may lead to missing any components during testing which makes bugs in the system under test. Their approach arise problem when the system under test does not match the system model.

6.4 Generating Test Cases by Novel Technique

Baikunda Narayan Baiswal et al [2] describe a test case generation approach which generates test cases by analyzing sequence diagrams and class diagrams of scenario, which achieves maximum path coverage criteria. Here they propose an approach called TC-ASEC approach, which generate test cases from UML design models using activity, sequence and class diagrams. They use gray box testing method which combines both black box testing and white box testing. They have covered path coverage criteria which have highest priority among all the coverage criteria for testing. Their approach is not fully automated. Since it supports reuse and defect analysis, their approach make more changes early in the development phase.

6.5 Generating Test Cases Using Event Based Approach

Sandeep Kumar Singh et al [19] presents an event based approach using events to generate test cases. Software models describe desired behavior of the system under test (SUT). Events taking place in SUT are proposed as Event Templates. The Event Flow Model is built from these Event Templates. This model is represented as Event Graphs. Their approach is very effective for real time systems since it is more logical rather than UML models. Their approach is independent of UML models and eliminates its necessity. The drawback here is linking

connections to the events is difficult. The event flow model construction is time consuming than UML models.

6.6 Why Genetic algorithm is efficient than other approaches?

M. Prasanna et al. [24] have proposed a method to automate the test case generation from sequence diagrams. The approach generates test cases for a sequence diagram of one real time application using Rational Rose Tool automatically. The test case generator converts all functionalities of the sequence diagram into a tree structure. Here, Genetic algorithm has also been discussed. Depth first search algorithm is applied. Then, the crossover technique is applied on the tree structures to identify the sequences. From the sequences, valid, invalid and terminate sequences are identified. The effectiveness of the testing is done by fault injection technique. The Methodology includes the following steps: (1) Analyze the Real system to be tested, (2) Construct sequence diagram using tool, (3) Convert model into tree structure, (4) Apply Depth first search and generate valid sequences, (5) Apply Crossover technique on all sequences, (6) Finally valid, invalid, and terminated test sequences are obtained from the new generation test sequences. This provides optimal way of generating test sequences from sequence diagram. This method achieves better and reliable test cases for object oriented applications.

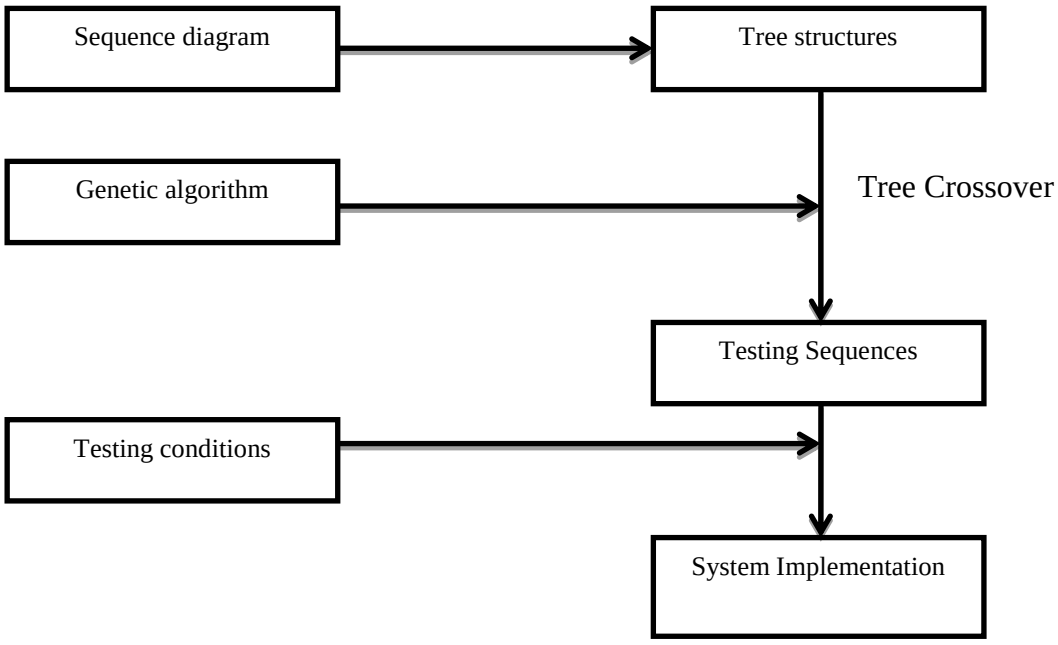

Figure 6: M. Prasanna et al. [21] Flowchart of proposed methodology

7. Discussion

Karambir et al [9] describes different techniques involved in generating test case generation by analyzing the dynamic behavior of objects. The scope of their paper is to study the techniques of test case generation using genetic algorithm, test case generation by random based testing, test case generation using combination of activity and sequence diagram and test case generation using model based testing. First, test case generation using genetic algorithm are derived by analyzing the dynamic behavior of object oriented software. In order to propose test cases from the object diagram, Genetic algorithm tree cross over should be derived. By applying the genetic algorithm in the object diagram, test cases are get generated. Second, test case generation using random based testing which generates test cases by analyzing the application and making assumptions. Initially it combines a pool of objects into a set or a list. Then, it creates a method for those sets. Test will be done by selecting any set or list under the method and it executes the testing method. This can be continued until all sets assumed for that application gets tested under the testing method. In this way, the test cases are derived using random based testing. Third, test case generation by CPM (Category Partition Method) i.e., generating test case by combining both activity and sequence diagram. Their technique helps to reuse the design. It generates test scenarios from activity and sequence diagram which achieves path coverage criteria perfectly. Following this, the test cases are generated by analyzing the respective scenario. Finally, test case generation by model based testing technique is described, where it uses the black box testing approach. Model based testing automates the complete design of test cases and generate traceability matrix, which traces link between requirements and generated test cases. Instead of writing hundreds of test cases, the test designer constructs an abstract model of the system under test. The MBT tool is used to generate a set of test cases from that model.

Li Bao Lin et al [10] describes a new test case approach where test cases are generated from UML sequence diagrams and Object Constraint Language (OCL). In their approach, a tree representation of sequence diagram is constructed. The conditional predicates are obtained by the traversal of the scenario tree. The conditional predicates are transformed into test data by applying function minimization technique. The test case generation process step include: (1) Converting sequence diagram into Scenario Tree, (2) According to the message path, distill the classes, attributes and operations, (3) Construct CCT(Class and Constraint Tuple) from OCL expressions, determine input data and generate test cases and (4) execute the test cases an validate the test results. The advantage of this approach is, it generates effective test cases which

achieves path coverage, pre- and post- condition coverage and link coverage. The cons here are it generates more test cases since it achieves more coverage criteria.

Santosh Kumar Swain et al [20] describes test case generation from Use case Dependency Graph (UDG) for use case diagrams and Concurrent Control Flow Graph (CCFG) for sequence diagrams. Their paper focuses on testing sequence of messages among objects of use case scenarios. The strategy derives test cases using full predicate coverage criteria. Some of the test adequacy criteria are also discussed. All-message-criteria (AMC) ensure all the messages between any two objects are tested at least once. All-path-coverage criteria (APC) ensure whether all the message paths are covered from start to end. Full-Predicate-Coverage criteria (FPC) ensure whether all possible combinations of the different predicates in the condition are checked. Condition Coverage Criteria (CC) covers all conditions in the graph. It includes all conditions as well as all loops. Branch-Coverage Criteria (BC) checks whether the set of test cases ensures at least one path covers the condition. Iteration Coverage Criteria (IC) requires all iterations to be covered once, twice and K times. The proposed approach derives use case dependency scenarios, derives test requirements from sequence diagrams and generates test cases for integration and system testing. The drawback here is, complexity of generating final test cases from sequences of use cases derived from sequence diagrams.

Emanuela G. Cartaxo et al [4] describes the procedure about generating test cases from sequence diagram translated into labeled transition diagram. The systematic procedure is presented for functional test case generation for feature testing of mobile phone applications. Their procedure includes: (i) Obtaining Labeled Transition System (LTS) model for functional testing and (ii) Deriving test cases. For deriving test cases it is necessary to identify all paths from LTS. A path table can be derived by traversing the LTS model using Depth First Search Algorithm. The Condition Coverage for the systematic procedure is covered. Each path in that path table results in a test case. Their procedure become complex if the number of loops exceeds.

Philip Samuel et al [16] describe test case generation using dynamic slicing approach. A message dependency graph is created from UML sequence diagram. Dynamic slicing approach is applied on MDG (Message Dependency Graph). Based on slice created from each message predicate, test cases are generated. For creating dynamic slices an Edge Marking method is used here. Their edge marking method uses program dependency graph. Edge marking algorithm is based on marking and unmarking the unstable edges appropriately as and when dependencies

arise and cease at run time. Their approach generates test case which satisfies all constraints corresponding to a slice. They have implemented a tool for automated test case generation namely UTG (UML Test case Generator). UTG has been implemented using Java and can easily integrate with any UML CASE tools like Magic Draw UML that supports XML (Extensible Markup Language) format. Since UTG takes UML models in XML format as input, UTG is independent of any specific CASE tool. The Magic Draw UML tool is used in several UML designs which found effective in generating test cases. The generated test cases were found to achieve the desired coverage. This is platform dependent and Tool dependent.

Baikunda Narayan Biswal et al [2] describes about test case generation from activity, sequence and class diagram of the scenarios, which achieves maximum path coverage criteria. Here, the proposed test case generation approach is named as TC-ASEC approach. In their proposed scheme, they use the method of gray box testing where it combines both white box testing and black box testing methods. Activity diagram gets parsed, where the test scenarios which satisfy path coverage criteria alone generates test cases. Among all other criteria, path coverage criteria have high priority which has been described here. After all test scenarios are generated test cases, class and sequence diagrams are generated from the test scenarios. Now by using Category Partition Method, analyze the system into functional requirements. For each functional unit of the system, its characteristics and parameters are identified. Then the test cases are generated based on its characteristics and parameters. Here the complete approach is done using JAVA Swing and Rational Rose software by implementing ATM case study. The approach is platform dependent and tool dependent.

A.V.K. Shanthi et al [1] focus on test case generation by means of using Genetic Algorithm. The first step is to extract information from the sequence diagram of the given software design. For that a parser program in java is used to extract essential information from the sequence diagram. From the extracted information, a SDT (Sequence Dependency Table) is constructed. Using SDT, test path is generated. By applying Genetic algorithm, test cases are get generated. Their approach generates best optimized test cases and validates test results by prioritization.

Shanmuga Priya et al [20] describe a model based testing approach from which the test paths are automated and obtained before development process. The test cases are executed in fixing errors. The automated test paths will give an idea to a software developer for ensuring

whether all the paths are covered and coded properly. The proposed model based testing approach undergoes these steps: (i) Requirements Analysis, (ii) Design using UML Sequence Diagram, (iii) Generate Sequence Dependency Table, (iv) Generate Sequence Dependency Graph and (v) derive test paths. Their approach helps in generating test cases by tester during testing phase to check whether the system works as it is intended.

Monalisa Sharma et al [14] describe the steps involved in the process of transformation of Sequence Diagrams (SD) into Sequence Diagram Graph (SDG) and then generating test cases from SDG. To create a SDG for any sequence diagram, identify the set of all operational scenarios. For each operational scenario, identify the set of all events. SDG contains the following data: attributes of the objects in that state, arguments in that state, range of values of all attributes in that state. Information that is obtained from data dictionary also stored in SDG. Finally, SDG is constructed considering the operational scenarios and set of data that are given as input. By covering all the paths from the start state to the end state in SDG graph, eventually covering all message paths as well as all interactions derive a test set. The test set thus generated achieves Path Coverage criteria. The generated test cases are based on event flow of operational scenarios. Generating event flows is very complex when operational scenarios are more.

Monalisha Khandai et al [13] describes about a novel approach of test case generation for concurrent systems using UML Sequence Diagrams for ATM case study. Their approach consists of transferring Sequence Diagram into Concurrent Composite Graph (CCG). The CCG is traversed using graph traversing algorithms like Breadth First Search and Depth First Search algorithm to generate test cases for concurrent systems. Their approach is easier for ATM case study. It is not applicable for larger projects or any other systems.

Saswat Anand et al [21] present an orchestrated survey of the techniques for automated test case generation of software systems. The survey includes techniques like test case generation by: (i) structural testing using symbolic execution, (ii) model based testing, (iii) combinatorial testing, (iv) random based testing and (v) search based testing. Their paper describes about all testing methods and test case generation by the above techniques but does not prove which method is efficient or easiest method among all methods.

Jayanthy.S et al [7] have also describe about genetic algorithm in the work of pattern recognition. Their approach is to do power testing on crosstalk delay faults in VLSI circuits. This

technique deals with low power consumption and high fault coverage criteria. Automatic test pattern generation approach is proposed to test crosstalk delay faults that affects the timing behavior of the circuits.

Liping Li et al [11] propose an automatic approach to generating test cases from UML activity diagrams based on Extension Theory. Through their approach the test cases generated not only satisfies test coverage criteria but also the number of test cases is decreased. Extenics is a new discipline to deal with contradiction problems with formalizing model and making novel decisions. Extenics theory when combines with engineering field solves more contradictory problems in various fields of information technology, automation, design and management, etc. To generate test cases from activity diagram by Extenics based theory, draw an activity diagram and refine XML file from the activity diagram. Construct Euler circuit for the activity diagram. Euler circuit is the most used graph problem in graph theory. Here the proposed method generates test cases from Euler circuit using Euler Circuit Algorithm. Constructing Euler circuit is complex and generating test cases from the algorithm is also complex. The advantage here is, it deals with contradictory problems and the number of test cases got decreased.

Bhuvaneswari.M.C.et al [3] describes about testing of asynchronous circuits and synchronous circuits. Their approach presents that the test pattern generation from asynchronous circuit systems. The paper describes the abstraction of finite state synchronous vectors from the circuits which can be applied for testing.

Wan-Hui Tseng et al [25] describes a model based scenario test case generation for nuclear safety systems. This was achieved by converting the scenarios described in natural language in a Safety Analysis Report to UML Sequence Diagrams based on proposed ontology design. Then, extract the initial environmental parameters and describe the operational sequences. Then perform variation on these systematic data to generate test cases. Their technique is completely applicable for real time systems.

8. Conclusion

The automated test case generation from UML Sequence diagrams using GA algorithm and several techniques has been reviewed. After analyzing, test case generation using Evolutionary Algorithms like Genetic algorithm is effective rather than any other technique. Since, the method of mutation analysis, evolutionary algorithms like Genetic Algorithm have

yielded 80.3% effectiveness in generating test cases. According to M. Prasanna et al. [21] GA provides optimal way of generating test sequences from sequence diagram. This method achieves better and reliable test cases for object oriented applications. Researchers use model based approach in which genetic algorithm's crossover technique is applied. But test case generation using other techniques does not apply fault injection technique. Since the use of evolutionary algorithms yields effectiveness in automated test case generation, it serves efficient than other approaches.

9. References

[1] A.V.K. Shanthi, Mohan Kumar, "Automated Test Cases From UML Sequence Diagrams", International Conference on Software and Computing Applications, Vol. 41, IACSIT Press, Singapore, 2012.

[2] Baikuntha Narayan Biswal, Pragyan Nanda, Durga Prasad Mohapatra, "A Novel Approach for Scenario Based Test Case Generation", International Conference on Information Technology, National Institute of Technology, IEEE Publication, Pages 244 - 247, 2009.

[3] Bhuvaneshwari. M. C, Sivanandam. N. C, "Testing asynchronous sequential circuits using synchronous circuit model", AMSE Journal, Modelling A, Vol.78, Issue 2, Page. 63,2005.

[4] Emanuela G. Cartaxo, Francisco G. O. Neto and Patricia D. L. Machado, "Test Case Generation by means of UML Sequence Diagrams and Labeled Transition Systems", IEEE Journal of Research, IEEE Publication, Pages 1292-1297, 2007.

[5] Franz Rothlauf**,** "Representations for Genetic and Evolutionary Algorithms 2nd Edition", Springer Journal, 2010.

[6] Hirohide Haga, Akihisa Suihiro, "Automatic Test Case Generation based on Genetic Algorithm and Mutation Analysis", IEEE International Conference on Control System, Computing and Engineering, Pages 119-123, Penang, November 2012.

[7] Jayanthy.S, Bhuvaneswari.M.C, "An efficient multi-objective genetic algorithm for low power testing of crosstalk delay faults in VLSI circuits", AMSE Journal, Advances B, Vol.54, Issue 2, Page 28, 2011.

[8] Jim Arlow, Ila Neustadt, "UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design (2nd Edition)", Pages 77-81, 2005.

[9] Karambir, Kuldeep Kaur, "Survey of Software Test Case Generation Techniques", International Journal of Advanced Research in Computer Science and Software Engineering, Vol. 3, Issue 6, June 2013.

[10] Li Bao-Lin, Li Zhi-shu, Li Qing, Chen Yan Hong, "Test Case automate Generation from UML Sequence diagram and OCL Expression", International Conference on Computational Intelligence and Security, IEEE Publication, Pages 1048-1052, Harbin, China, December, 2007.

[11] Liping Li, Xingsen Li, Tao He, Jie Xiong, "Extenics-based Test Case Generation for UML Activity Diagram", Elsevier Journal, Information Technology and Quantitative Management, Pages 1183-1193, 2013.

[12] M. Prasanna, K.R. Chandran, "Automatic Test Case Generation for UML Object diagrams using Genetic Algorithm", International Journal of Advance Soft Computing Applications", Vol.1, No.1, Pages 19-32, July 2009.

[13] Monalisa Khandai, Arup Abhinna Acharya, Durga Prasad Mohapatra, "A Novel Approach of Test Case Generation for Concurrent Systems Using UML Sequence Diagram", IEEE 3rd International Conference on Electronics Computer Technology , Pages 157-161, Kanyakumari, India, April, 2011.

[14] Monalisa Sarma, Debashish Kundu, Rajib Mall, "Automated Test Case Generation from UML Sequence Diagrams", 15th International Conference on Advanced Computing and Communications, IEEE Computer Society, IEEE Publication, pages 60-67, Guwahati, Assam, December, 2007.

[15] Paramjit Kaur, Rupinder Kaur, “Approaches for Generating Test Cases Automatically to Test the Software”, International Journal of Engineering and Advanced Technology, Vol.2, Issue 3, February 2013.

[16] Philip Samuel, Rajib Mall, “A Novel Test Case Design Technique Using Dynamic Slicing of UML Sequence Diagrams”, e-Informatica Software Engineering Journal, Vol.2, Issue 1, 2008.

[17] Ranjita Kumari Swain, Vikas Panthi, Prafulla Kumar Behera, “Test Case Design Using Slicing of UML Interaction Diagram”, 2nd International Conference on Communication, Computing and Security, Vol.6, Pages. Pages 136 - 144, November, 2012.

[18] Sangeeta Sabharwal, Ritu Sibal, Chayanika Sharma, “Applying Genetic Algorithm for Prioritization of Test Case Scenarios Derived from UML Diagrams”, International Journal of Computer Science Issues, Vol. 8, Issue 3, No. 2, May 2011.

[19] Sangeeta Sabharwal, Sandeep Kumar Singh, Dhruv Sabharwal, Aditya Gabrani, “An Event Based Approach to generate test scenarios”, International Conference on Computer and Communication Technology, IEEE Publication, Pages 551-556, Allahabad, Uttar Pradesh, September, 2010.

[20] Santosh Kumar Swain, Durga Prasad Mohapatra, Rajib Mall, “Test Case Generation Based on Use case and Sequence Diagram”, International Journal of Software Engineering, Vol. 3, Issue 2, July 2010, Pages 21 - 52, July 2010.

[21] Saswat Anand, Edmund K. Bruke, Tsong Yueh Chen, John Clark, Myra B. Cohen, Wolfgang Grieskamp, Mark Harman, Mary Jean Harrold, Phil Mccinn, “An orchestrated survey of methodologies for automated software test case Generation”, Elsevier Journal, The Journal of Systems and Software, Vol. 86, Issue 8, Pages 1978-2001, February 2013.

[22] S. Shanmuga Priya, P.D. Sheba Kezia Malarchelvi, “Test Path Generation using UML Sequence Diagram”, International Journal of Advanced Research in Computer Science and Software Engineering, Vol. 3, Issue 4, pp. 1069 - 1076, April 2013.

[23] Venkatesan. R, Sivanandam. S.N, Prasanna. M, "A Model based approach for Test case generation in object oriented systems", AMSE Journals, Advances D, Vol. 13, Issue 1, Page 18, 2008.

[24] V. Mary Sumalatha, G.S.V.P. Raju, "Object Oriented Test Case Generation Technique using Genetic Algorithms", International Journal of Computer Applications, Vol. 61, Issue 20, January 2013.

[25] Wan-Hui Tseng, Chin-Feng Fan, "Systematic scenario test case generation for nuclear safety systems", Elsevier Journal, Information and Software Technology, Vol. 55, Issue 2, Pages 344-356, February 2013.

[26] William E. Perry, "Effective Methods for Software Testing Includes Complete Guidelines and Checklists", 3rd edition, Wiley Publication.

[27] Software Testing, Wikipedia, http://en.wikipedia.org/wiki/Software_testing, 2014.

[28] Test data Generation, Wikipedia, http://en.wikipedia.org/wiki/Test_data_generation,2014.

[29] Genetic algorithm, Wikipedia, http://en.wikipedia.org/wiki/Genetic_algorithm, 2014.