Volume 3, Issue 6, June 2013 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Survey of Software Test Case Generation Techniques

Karambir
Assistant Professor
UIET Department of computer Sc. & Engg, Kurukshetra University Kurukshetra , India

Kuldeep Kaur
Research Scholar
UIET Department of Computer Sc. & Engg, Kurukshetra University Kurukshetra, India

Abstract: Software testing is an crucial part of software development which guaranteed the verification and validation process of the software. In order to do software testing we must have to apply the method of mapping the software for all its transition states and individually validating the output for a set of given input. For a any given part of software we will be writing a set of test cases that called test suites and it is used to group together similar test cases. Test suites is a collection of test cases that are planned to be used to test a software program to illustrate that it has some specific set of behaviors. In order to find out how a test case is valid or not for that we do not have specific mechanism. We mostly depend on the software testers understanding of the requirement. The scope of this paper to study different technique use in test cases, for example test case generation using genetic algorithm, test case generation using random based testing, test case generation using Model based testing. The test cases are derived by analyzing the dynamic behavior of the objects due to external and internal stimuli.

Keywords : Software Testing ,Test case ,Genetic algorithm, Model based testing, Random Testing.

I. INTRODUCTION

Software testing is an important activity in software development life cycle. Software organizations spend large portion of their budget in testing related activities. Software Testing is a costly and time consuming process in software development life cycle. 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. The Automation of testing will not be that much productive in terms of cost and time consuming because if we have to wait till the end of the Software Development Life Cycle stage (SDLC). Software Testing[1] is any process or activity aimed at evaluating a system or attribute or capability of a program and determining through the purpose to find that whether it satisfies or meets the specified requirements or not. In simple words testing is executing a system in order to discover any errors, gaps or missing requirements in contrary to the actual requirements desire output.

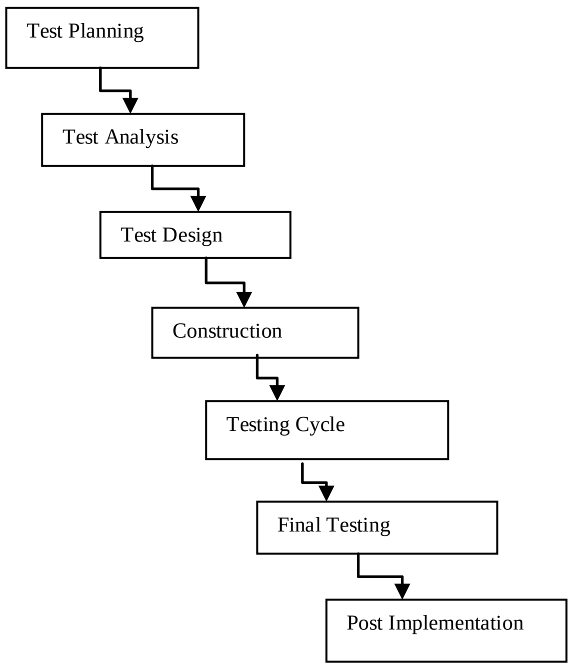

Figure1: Software Testing Steps

Test cases help the user to pen down entire coverage to the application and test all possible combinations in the application. It also provides the user to easily reproduce the steps that were undertaken to uncover a defect that as detected during test. It also provides the extent of which the testing has concluded and the areas in which the application is working fine. Throughout the years a huge number of different technique have been proposed for generating test cases. A test case is basically a description of a test. A Test case has the components that describe an input, event or action and expected response to determine if the feature of an application is working correctly. Test cases can be mapped directly to. Test cases are derived from use cases and can also be derived from system requirements. The one of the main advantages of generating test cases from requirements specifications and design will often help the software or test engineer to discover problems as early. As they can be created earlier in the development life cycle and get ready for use before the programs are constructed. Generating test cases early helps Software Engineers or test engineer can often find ambiguities and inconsistencies in the requirements specification and design documents. This will absolutely take down the cost of building the software systems as errors are eliminated early during the life cycle. The below diagram[2] highlights the steps of software testing.

II. Related Work

Rudnick Elizabeth M. and Greenstein Gary S.[3] described test generation using fault-oriented algorithms was highly complex and time consuming. Genetic Algorithms were first used as a framework for the simulation-based test generation. GAs are composed of populations of strings or chromosomes and three evolutionary operators: Selection, Crossover and Mutation. The CRIS test generator used as logic simulator to evaluate candidate test sequences, The test sets generated often had lower fault coverage than those generated by a deterministic test generator. Cavarra *et al.*[4] described a modeling architecture for the purposes of model based verification and testing. Their structural design contains two components. The first component of the architecture was the system model that was written in UML; this was a collection of class, state and object diagrams: the class diagram identified the entities in the system, the state diagrams gave justification how these entities might evolved, the object diagram specified an initial configuration. The second component was also written in UML, was the test directive. This test directive consists of particular state and object diagrams, the object diagrams were used to state coverage criteria and test constraints ; the state diagrams were used to identified test purposes. The test directives and system model be able build using any of the standard toolsets like Rational Rose.

T.Y. Chen *et al.* [5] developed a choice relation framework for supporting category-partition test case generation. All the constraints among choices must be defined manually. They captured the constraints among choices in rigorous and systematic manner via the introduction of various relations. The framework included the following features like consistency checks of specified constraints among choices, automatic deductions of new constraints among choices whenever possible, and a more effective test frame construction process. Category Partition Method(CPM) was a specification-based testing method that helps the software testers to formed test cases through refining the functional specification of a program into test specifications. They enable the software tester to specified the relative priorities for choices that were used for the subsequent formation of complete test frames. These test frames can subsequently be used as the basis for generating test cases. They applied their approach to real-life situations and reported on the efficiency of consistency checks and automatic deductions of choice relations. Franck Fleurey [6] presented a complete and automated chain for test cases derivation from formalized requirements in the perspective of object-oriented embedded software. However to automate the test generation process, there was huge space to bridge between high level use cases and concrete test cases. The test cases are generated into 2 steps. Use cases orderings are deduce from use case contracts, and that use case scenarios are substituted for each use case to generate test cases. Their purpose was to generate test cases for powerfully detecting faults in embedded s/w and is to cover the system in terms of statement coverage with those generated tests.

Monalisa sarma [7] presented a approach of generating test cases from UML design diagrams. Having stored all essential information for test generation in the STG, they now traverse the STG to generate test cases. The TestSuiteGeneration algorithm traverse the STG at 2 levels. The traversal begins with the UDG. This traversal visits all use cases and generate test cases for detecting initialization faults. At level 1, If a use case initialization faults occur then it was assume faults in its operation and therefore no need to apply test cases corresponding to the operation. At level 2 traversal, starting from a use case node the corresponding SDG was visited and test cases were generated to detect operational faults. Alexander pretschner *et al.* [8] developed test case generator "AutoTest" to generate and run random tests for 27 classes from a usually used Eiffel library. The random generation of test input data was attractive because it was generally appropriate and cheap, both in conditions of implementation effort and execution time. It tested all its methods, when AutoTest start to tests a class. They presented the result of an empirical study that was based on randomly testing 27 Eiffel classes in 1215 hours .Each with 30 seeds of the random number generator. Analyzed over 6 million failures triggered during the experiments, the study provides proof that random testing was predictable in terms of the relative number of defects detected over time. Hassan Reza *et al.*[10] discussed a model based software testing method for web applications that utilizes behavioral models of the SUT from Statechart models originally devised by Harel (1987,1988). Statecharts model can be used for both for modeling and generating test cases for a web application. The main focus of their work was on the front end design and testing of web applications. As such, they utilizes the syntax of web pages to guide the specification of the Statcharts. Their approach was a systematic to test the front-end functionality of web application. For most parts, they are concerned with verifying that the links, form and images in the web application under test function according to specification documents. Furthermore, they address to how to model the web applications with the Statchart in their work. To generate test cases from Statchart diagram, they defined five coverage criteria: (1) all-blobs (2) all- transitions (3) all- transitions -pairs (4) all-conditions (5) all-paths. Chandran and Prasanna M.[11] proposed a model to generate test cases for the object diagrams. This methodology various steps were evolved. Object diagrams were constructed, stored and object are named. The tree was build using these object names which further applied with crossover operator of GA. These new generation trees are converted into binary tree and traversed using DFS technique and this gave all the valid, invalid and termination sequences for a given application. Mutation analysis was done by infecting faults into the system are then results are compared with original, if there exists some differences then these mutants are considered as killed test.

Kulvinder Singh *et al.* [12] stated that genetic algorithm could be widely used as optimization techniques. Some of the researchers concluded that genetic algorithm has important impact on the optimize test cases generation. The conventional methods did not given fault free software in short time. It took them long period of time. So there was need of automated test case generation using genetic algorithm which could found large vector of faults in short time. This was demonstrated with the help of an example in the paper.

III. Testing Stages

Test cases help the user to pen down entire coverage to the application and test all possible combinations in the application. It also provides the user to easily reproduce the steps that were undertaken to uncover a defect that as detected during test. It also provides the extent of which the testing has concluded and the areas in which the application is working fine. Through the years a number of different methods have been proposed for generating test cases. A test case is a explanation of a test, independent of the way a given system is designed. Test cases can be mapped directly and derived from use cases. Test cases can also be derived from system requirements.. Additionally, when the test cases are generated early, Software Engineers can often find ambiguities and inconsistencies in the requirements specification and design documents. This will definitely get down the cost of building the software systems as errors are eliminated early during the life cycle. The below diagram[13] highlights the steps of software testing.

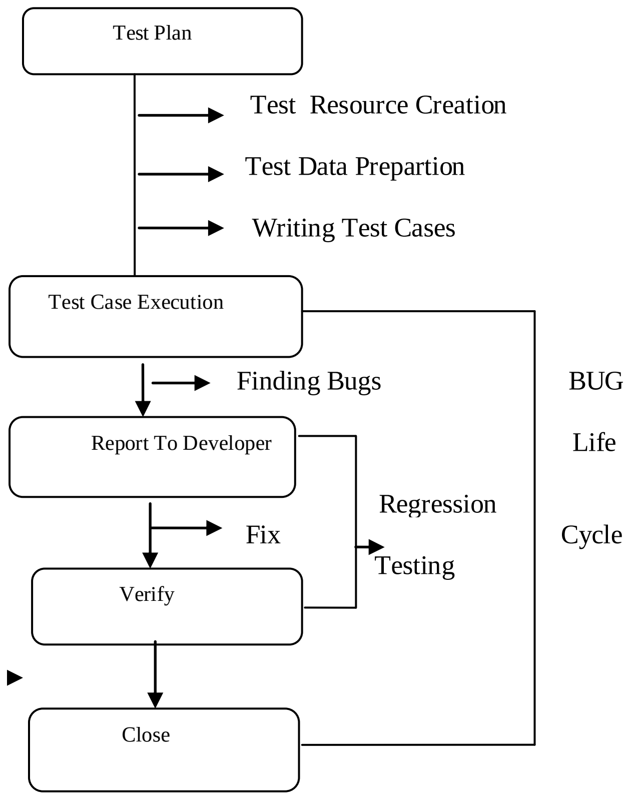

Figure 2 : Software Testing Stages

The above steps are under taken to test an application or software . Firstly the test plan is created which includes the overall structure of the plan i.e. requirements for testing, testing flow etc. Then the test cases are written according to some specific technique. There are many existing techniques are available. These cases are executed under suitable environment. The bugs are found out and reported to the developers. The developer fixes these bugs and finally these are verified using some testing technique. This way the whole procedure works.

IV: Software Testing Technique

Technique1: Test Case generation using Genetic Algorithms (GA)

In this technique, automated generation of test cases in object oriented systems has been presented. The test cases are derived by analyzing the dynamic behaviour of the objects due to internal and external stimuli. The study is limited to the object diagrams taken from the UML (Unified Modelling Language) model of the system. In order to carry out all suitable test cases of a given object diagram, Genetic Algorithm's (GA) tree crossover has been proposed.

Technique2: Test Case generation using Random Testing :

Random test case generation is a technique where in the test cases are generated not based on an algorithm but based on the ones assumption of the application. This technique is implemented on the above mentioned case study. The following classes will be tested and various test inputs will be provided to check for the faults. The framework used here to validate is called AutoTest and using this we will be able to predict the number of issues found and the number of detects undetected. The AutoTest framework classifies test cases into the following categories: passed (no exception), unresolved (precondition violation in method under test), or failed (other exception).

Method create test depicted below contains the main loop of the testing strategy used in this technique. At each step it selects a method for testing and then causes the execution of this method.

```
create_test ( timeout ) :
 from
 initialize pool
 until timeout
   loop
   m := choose ( methods under test () )
   create_test for method (m)
    end
```

1. Method Initialize pool creates an empty pool of objects to be used for testing .
2. The Method methods under test returns the set of methods under test.
3. The non-deterministic method choose selects an arbitrary element of a set or a list.
4. The method create_test for method is responsible for generating a call to method m.

Technique3: Test Case generation using a combination of Activity Diagrams and Sequence diagrams

Category Partition Method is simply a specification based testing technique with respect to some specific criteria. CPM first decomposes the functional specification into functional units and then examines each functional unit. It finds the categories for each parameter and environmental condition. It helps in identifying the parameters and environmental conditions that affect the execution behavior of the function. CPM is also helpful in finding the categories of information that characterize each parameter and environmental condition. We will first generate test scenarios from activity diagrams, which achieve path coverage criteria perfectly, followed by generation of test cases by analyzing the respective sequence and class diagrams of each scenario. This technique helps to reuse the design. Complex tests are built up by designing a test that runs through a series of use cases. First the test scenarios will be generated followed by analysis of each scenario using the sequence diagrams to formulate the test cases. Following gives a pictorial representation of the approach followed.

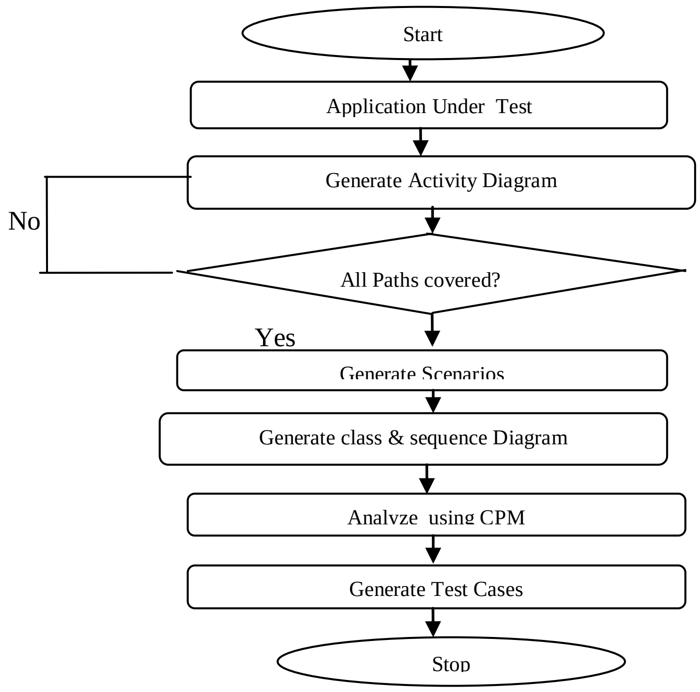

Figure 3. Pictorial Representation of Test scenario

Technique4: Test Case generation using Model Based Testing

Model Based Testing (MBT) is a black-box testing technique where common testing tasks such as test case generation and test result evaluation are automated based on a model of the application under test. This approach has recently spread to a variety of software domains but originates from hardware testing, most notably from telephone switches, and from the increasing use of object orientation and models in software design and software development.

Model Based Testing (MBT) automate the complete design of test cases and the generation of the traceability matrix, which traces the 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. Below describes the manner in which the MBT takes place.

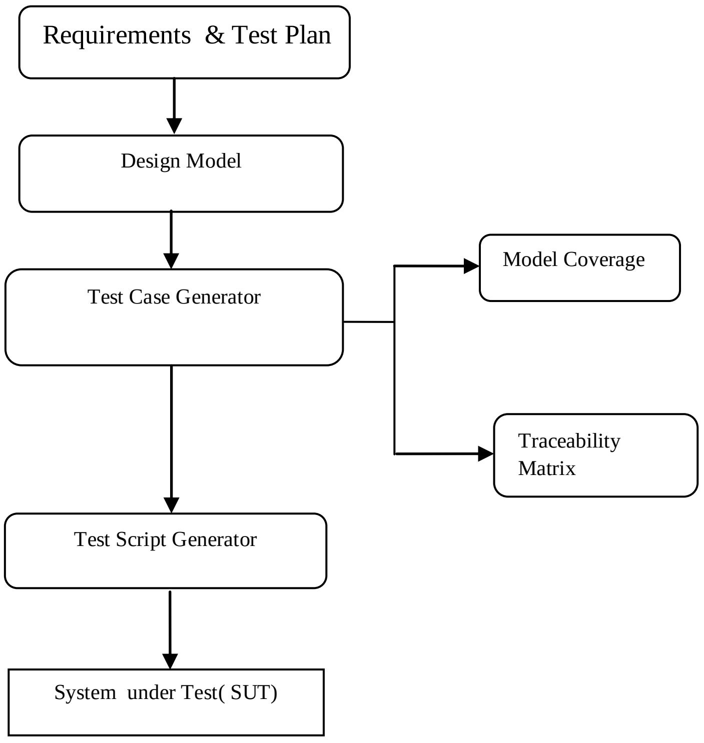

Figure4. Model based Testing Process Diagram

1. The first step of model-based testing[11] is to write an abstract model of the system that we want to test. After writing the model, it is advisable to use tools to check that the model is consistent and has the desired behaviour.
2. The second step of model-based testing is to generate abstracts tests from the model. We must choose some test selection criteria, to say which tests case we want to generate from the model because there are usually infinite number of possible tests. The main output of this step is a set of abstracts tests, which are sequences of operations from the model. Model based-testing tools produce a requirements traceability matrix or various other coverage reports as additional outputs of this step. The Traceability matrix traces the link between functional requirements and generated test cases. This is basically a many-to-many relation: a requirement can be covered by several test cases and a single test case may exercise several requirements. The coverage reports give us some indications of how well the generated test set exercises all the behaviours of the model. We can use such coverage reports simply for statistical feedback about the quality of the generated test set or we can use them to identify parts of the model that may not be well tested and investigate why this has happened.
3. The third step of model-based testing is to transform the abstract tests into executable concrete tests. this may be done by a transformation tool, which uses various templates and mappings to translate each abstract test case into an executable test script.
4. The fourth step is to execute the concrete tests on the system under test. With online model-based testing, the tests will be executed as they are produced.

V. Conclusion

We generate various test cases technique directly from UML behavioral diagram, where the design is reused. By using our approach defects in the design model can be detected during the analysis of the model itself. So, the defects can be removed as early as possible, thus reducing the cost of defect removal. First we generate test case generation using genetic algorithm, random based testing, model based testing and test scenarios from the activity diagram and then for each scenario the corresponding sequence diagram generated .After analyzing each category, its significant values and constraints are generated and respective test cases are derived. Test coverage criteria achieved is another advantage of our approach.

References

[1]. www.tutorialspoint.com/software_testing.
[2]. http://www.buzzle.com/articles/phases-of-testing-life-cycle.html.
[3]. Rudnick Elizabeth M., Greenstein Gary S., “A Genetic Algorithm Framework for Test Generation” IEEE Transactions on Computer Aided design of Integrated Circuits and Systems, VOL. 16, NO. 9, 1997.
[4]. Cavarra, A., C. Crichton, J. Davies, A. Hartman, T. Jeron and L. Mounier. Using UML for automatic test generation. Oxford University Computing Laboratory, Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2000.
[5]. T.Y. Chen, Pak-Lok Poon and T.H Tse," A choice Relation Framework for supporting category-partition Test case Generation", IEEE Transactions on software engineering Vol. 29, No.7, 2003.
[6]. Clementine Nebut, Franck Fleurey, Yves Le Traon and Jean-Marc Jeze quel, "Automatic Test Generation: A Use Case Driven Approach", IEEE Transactions on software engineering Vol.32, No.3, 2006.
[7]. Sarma Monalisa and Mall Rajib, “Automatic Test Case Generation from UML Models,” 10th International Conference on Information Technology, pp. 196-201, 2007.
[8]. Ilinca Ciupa, Alexander Pretschner, Andreas Leitner, Manuel Oriol, Bertrand Meye: "On the Predictability of Random Tests for Object-Oriented Software" International Conference on Software Testing, Verification, and Validation, 2008.
[9]. Reza, H., K. Ogaard and A. Malge," A model based testing technique to test web applications using statecharts", Proceedings of 5th International Conference on Information Technology: New Generations, pp. 183-188, 2008.
[10]. Prasanna M., Chandran K.R., “Automatic Test Case Generation for UML Object diagrams using Genetic Algorithm”,Vol.1,No.1,2009.
[11]. Singh Kulvinder and Kumar Rakesh, “Optimization of Functional Testing using Genetic Algorithms” International Journal of Innovation, Management and Technology, Vol. 1,No. 1, 2010.
[12]. Utting, Mark, and Bruno Legeard. Practical model-based testing: a tools approach. Morgan Kaufmann, 2010.
[13]. http://prajaktakanade.blogspot.in.