Advances in Computer Science and Information Technology (ACSIT)
p-ISSN: 2393-9907; e-ISSN: 2393-9915; Volume 3, Issue 3; April-June, 2016, pp. 182-186

http://www.krishisanskriti.org/Publication.html

A Review on Optimization Techniques for Test Case Generation

K P Yadav[1], Saroj Patel[2] and Tannu Arora[3]

[1]Director KCC Institute of Tech & Mgmt, Greater Noida
[2,3]Jodhpur National University, Jodhpur
E-mail: [1]drkpyadav732@gmail.com, [2]drsarojpatel@gmail.com, [3]tannu.arora@gmail.com

Abstract—*This paper summarizes a survey in the field of automated software testing. The design of an appropriate test suite for software testing is a challenging task. It requires a suitable tradeoff between effectiveness, e.g., a sufficient amount of test cases to satisfy the test goals of a given coverage criterion, and efficiency, e.g., a redundancy-reduced selection of test cases. Recent test suite optimization approaches, usually require an explicit enumeration of existing test cases to select from. The test suite design for covering entire software product is more problematic as dependency between test cases, test goals and product configurations has to be taken into account. Due to the exponential number of configurations with respect to the number of features, an explicit enumeration of all products for optimizing a product-line test suite is impartibly. There have been few efforts on representing a brief classification, which covers mostly used automatic test case generation approaches. In this paper we introduce a general classification for automatic test case generation approaches with comparison between these approaches to show that model based testing is the most acceptable approach to generate automatic test cases.*

Keywords*: Automated software testing; optimization approaches; automatic test case generation; model based testing*

1. INTRODUCTION

Automated software testing is a development in which software tools execute pre-scripted tests on a software application before it is released into production. Automated testing tools are able of executing tests, reporting conclusions and comparing results with previous test runs. Tests carried out with these tools can be run repeatedly, at any time of day. The method or process being used to put into practice automation is called a test automation framework. Several frameworks have been realized over the years by commercial vendors and testing organizations. Automating tests with commercial off-the-shelf (COTS) or open source software can be complex, however, because they almost always necessitate customization. In many organizations, automation is only put into practiced when it has been resolute that the manual testing program is not meeting prospect and it is not possible to bring in more human testers.

2. REVIEW ON AUTOMATED SOFTWARE TESTING

Researchers have proposed different techniques to generate test case automatically. Software testing is the process of executing a program in order to find faults. Testing is very important, though expensive phase in software development and maintenance; it has been estimated that software testing entails between 30 percent and 50 percent of software development. A test case is a set of tests performed in a sequence and related to a test objective, which will produce a number of tests comprising specific input values, observed output, expected output, and any other information needed for the test to run, such as environment prerequisites. There has been a significant amount of work in automatic test case generation that attempts to increase the amount of observed behavior. There are different reasons to automate test case generation task in software testing. Some of the most important reasons are as follows.

- Reducing the cost of software testing
- Reducing human errors
- Increasing software products quality
- Reducing number of test cases
- Covering all system requirements

2.1 Selection of Correct Test Cases for Automation Testing

Automation does not overpower or replaces manual testing but it compliments it. Like manual, automation too needs a strategy with proper planning, monitoring & control. Automation, when put into practiced correctly, can become an asset to the team, project and ultimately to the organization. There are many advantages of automation; here are few important to mention:

- useful to execute the routine tasks like smoke tests and regression tests.
- useful in preparing the test data.
- helps to execute the test cases which involve complex business logic.

- good to execute the cross platform test cases (like different OS, browsers etc.)
- great to execute the test cases which are a bit difficult to execute manually.
- when the number of iterations of the test case executions are not known.

Many a time stakeholders feel that test automation acts as a support tool for manual testing, so it's vital to recognize that automation is the best way to increase the effectiveness, efficiency and coverage of testing. it not only saves time but also improves accuracy as repetitive tasks via manual approach can prone to human errors and can be time consuming.

There are following steps to select test cases for automation of software testing process:

Step 1:

Identify the parameters on which will be based on test case as a candidate for automation.

- Test case executed with different set of data
- Test case executed with different browser
- Test case executed with different environment
- Test case executed with complex business logic
- Test case executed with different set of users
- Test case involves large amount of data
- Test case has any dependency
- Test case requires special data

Step 2:

Break each application into modules. for each module, analyze and try to identify the test cases which should be automated based on the parameters.

Step 3:

Consolidate and group the number of test cases for each module preprocessing.

3. GENERAL CLASSIFICATION OF OPTIMIZING TEST CASE TECHNIQUES

There has been a significant amount of work in automatic test case generation that attempts to increase the amount of observed behavior. Despite of these wide researches, there have been few efforts on representing an classification, which covers mostly used existing automatic test case generation approaches. Fig. 1 shows the general classification of optimizing techniques.

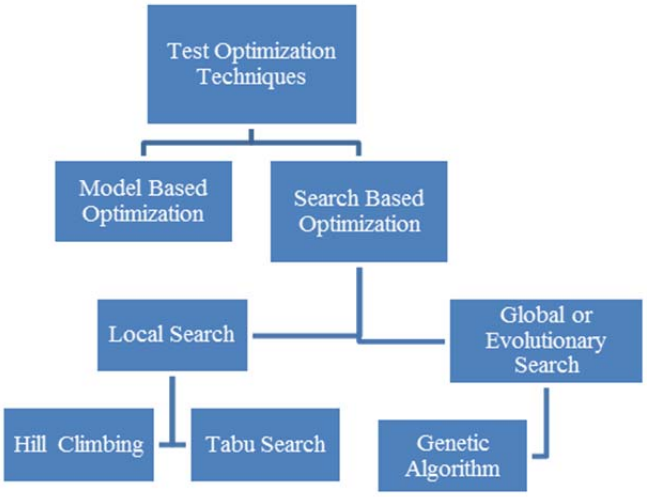

Fig. 1: Optimization Techniques

A comparison between these two approaches in terms of algorithm (see Table 1).

Table 1: Comparison between Optimizing Techniques

Approach	Real time Application	Complexity	Fault Analysis
Model Based	Yes	High	High
Search Based	Depends on Test Type	High	Medium

3.1 Model Based Testing

Testing is a necessary, but time and resource overwhelming activity in the software development process. Making a short, but effective test suite typically needs a lot of manual work and expert information. In a model-based process, among other subtasks, test building and test execution can also be partially automated. Model-based testing (MBT) aims at automated, scalable, and systematic testing solutions for complex industrial software systems[3].

Model-based testing is a software testing technique in which the test cases are resulting from a model that describes the functional features of the system under test. It makes use of a model to generate tests that includes both offline and online. The technical literature on model-based testing (MBT) offers us several techniques with different characteristics and goals[2]. Fig. 2 shows the basic components of MBT.

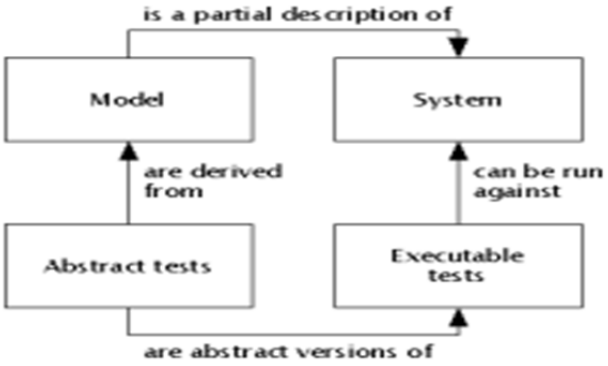

Fig. 2: Model Based Testing

Model-based testing is based on some form of a model that explains some aspects of the tested system in a way that enable automatic test generation. Modeled is either the system's desired behavior, on some level of abstraction, or alternatively testing strategies. When testing strategies are modeled, they are from time to time modeled as probabilistic processes. System models are usually not probabilistic[14].

Importance of Model-Based Testing

- Unit testing won't be enough to check the functionalities.
- To make sure that the system is behaving in the same sequence of actions.
- Model-based testing technique has been adopted as an included part of the testing process.
- Commercial tools are urbanized to support model-based testing.

3.2 Search-based[5,6] Automatic Test Case Generation

There are a variety of search algorithms to use for the purpose of test case generation. This paper uses evolutionary algorithms so we concentrate only on evolutionary algorithms. EA's describe the process of search, and allows us to carry out the search and they allow us to find optimal solution. Evolutionary algorithms

EA is a subset of evolutionary computation which is a subfield of Artificial intelligence. Evolutionary computation is a general term for several computational techniques. EC represents powerful search and optimization influenced by biological mechanisms of evolution. EAs refer to evolutionary computation models using randomness and genetic inspired operations. EAs involve selection, recombination, random variation and competition of the individuals in a population of adequately represented potential solutions. The potential solutions are referred as chromosomes or individuals. The evolutionary algorithms include Genetic Algorithms. Genetic algorithm represents the main paradigm of evolutionary computation.

3.2.1 Basics Of Genetic Algorithm (GA)[13]

The genetic algorithm is a method for solving both constrained and unconstrained optimization problems that is based on natural selection, the process that drives biological evolution. The genetic algorithm repeatedly modifies a population of individual solutions. at each step, the genetic algorithm selects individuals at random from the current population to be parents and uses them to produce the children for the next generation. over successive generations, the population "evolves" toward an optimal solution.

The genetic algorithm can apply to solve a variety of optimization problems that are not well suited for standard optimization algorithms, including problems in which the objective function is discontinuous, non differentiable, stochastic, or highly nonlinear. The genetic algorithm can address problems of mixed integer programming, where some components are restricted to be integer-valued. The genetic algorithm uses three main types of rules at each step to create the next generation from the current population:

- Selection rules select the individuals, called parents that contribute to the population at the next generation.
- Crossover rules combine two parents to form children for the next generation.
- Mutation rules apply random changes to individual parents to form children.

The genetic algorithm differs from a classical, derivative-based, optimization algorithm in two main ways, as summarized in Table 2[13].

Table 2: Comparison between Classical and Genetic Algorithm

Classical Algorithm	Genetic Algorithm
Generates a single point at each iteration. The sequence of points approaches an optimal solution.	Generates a population of points at each iteration. The best point in the population approaches an optimal solution.
Selects the next point in the sequence by a deterministic computation.	Selects the next population by computation which uses random number generators.

4. PERFORMANCE EVALUATION

Model based testing is gaining importance due to highest fault finding power. MBT approach generally generate large sets of test cases when applied to the real systems, regardless of the coverage criteria. All the test cases can be automated using model generated.

Genetic algorithms is also used for generating test case automation. These algorithms used genetic trees to optimize test cases from which fault analysis had been made using mutation testing. The effectiveness of test cases can be evaluated using a fault injection technique called mutation analysis.

Mutation testing[1] is a process by which faults are injected into the system to verify the efficiency of the test cases. Mutation-based analysis is a fault based testing strategy that starts with a program to be tested and makes numerous small

syntactic changes into the original program. Program with injected faults are inserted and tested in the respective manner. If a test case set is capable of causing behavioral differences between original program and mutant, mutant is considered as killed by test. The product of mutation analysis is a measure called mutation score, which indicated the percentage of mutants killed by a test set. Mutation analysis was conducted by inducing errors into the system to check for the fault analysis. Mutation analysis score was calculated as:

Mutation analysis score= No of fault found/no of fault injected

The percentage of mutation score had been calculated by performing different experiments. Experimental results showed that fault analysis done from genetic algorithms test case generation method is 80.3%while using model based testing score is 91.6% (see Fig. 3).

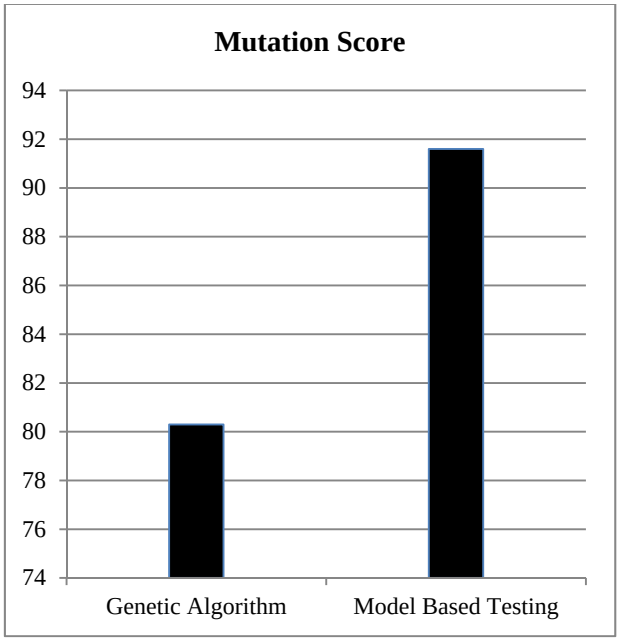

Fig. 3: Fault Analysis

From the experimental results, we conclude that model based methodology is useful to generate test cases. This technique although highly reliable and finds more faults, there are few drawbacks of this i.e., Designing the model is complex and needs skilled resources and the initial time taken to setup and understand the system to design the model is huge.

The issues can be overcome if we combine the GA technique with the MBT to give better results and robustness.

5. CONCLUSION

One critical task in software testing is to generate test cases. Since, test case generation has become an optimization problem hence scope remains open to apply some more techniques to achieve better results. Genetic algorithm for optimization of test cases has low fault finding power whereas, model based approach is not suitable to handle the large and complex system. This approach is very much suitable for simple systems where no more fork-joins, like nested-fork joins and etc. are involved, which is our next objective. However this approach is not sufficient to handle different kind of errors such as work flow errors, state based errors and etc. To overcome this bottleneck, a combined approach has to be develop for better optimization of test case generation. Although there have been lots of researches for optimization of software testing ,but for real time applications more researches are still needed.

REFERENCES

[1] MOGHADAM, M.H. ,BABAMIR, S.M.,MUTATION SCORE EVALUATION IN TERMS OF OBJECT-ORIENTED METRICS, IEEE 4TH INTERNATIONAL CONFERENCE ON COMPUTER AND KNOWLEDGE ENGINEERING,PP. 775-780,(2014)

[2] Arilo Claudio Dias-Neto, Guilherme Horta Travassos, "Supporting the Combined Selection of Model-Based Testing Techniques", IEEE Transactions on Software Engineering, vol.40, no. 10, pp. 1025-1041,(2014)

[3] Shaukat Ali, Muhammad Zohaib Iqbal, Andrea Arcuri, Lionel C. Briand, "Generating Test Data from OCL Constraints with Search Techniques", IEEE Transactions on Software Engineering, vol.39, no. 10, pp. 1376-1402(2013)

[4] L. Padgham, Zhiyong Zhang, J. Thangarajah, T. Miller, "Model-Based Test Oracle Generation for Automated Unit Testing of Agent Systems", IEEE Transactions on Software Engineering, vol.39, no. 9, pp. 1230-1244,(2013)

[5]M. Harman, A. Mansouri, and Y. Zhang, "Search Based Software Engineering: Trends, Techniques and Applications," ACM Computing Surveys, vol. 45, article 11,(2012)

[6]M. Phil, "Input Domain Reduction through Irrelevant Variable Removal and Its Effect on Local, Global, and Hybrid Search-Based Structural Test Data Generation," IEEE Trans. Software Eng., vol. 38, no. 2, pp. 453-477, (2012)

[7]S. Ali, L.C. Briand, and H. Hemmati, "Modeling Robustness Behavior Using Aspect-Oriented Modeling to Support Robustness Testing of Industrial Systems," Software and Systems Modeling, vol. 11, pp. 633-670, (2012)

[8] A.Arcuri, M.Z. Iqbal, and L. Briand, "Random Testing: Theoretical Results and Practical Implications," IEEE Trans. Software Eng., vol. 38, no. 2, pp. 258-277,(2012)

[9] C.D. Nguyen, S. Miles, A. Perini, P. Tonella, M. Harman, and M. Luck, "Evolutionary Testing of Autonomous Software Agents," Autonomous Agents and Multi-Agent Systems, vol. 25, no. 2, pp. 260-283, (2012)

[10] W.L. Andrade, D.R. Almeida, J.B. Candido, and P.D.L. Machado, "SYMBOLRT: A Tool for Symbolic Model-Based Test Case Generation for Real-Time Systems," Proc. 19th Tools Session of the Third Brazilian Conf. Software: Theory and Practice, pp. 31-37, (2012)

[11] Model Development Tools, http://www.eclipse.org/modeling/mdtproject=ocl,(2012)

[12] G. Fraser and A. Arcuri, "EvoSuite: Automatic Test Suite Generation for Object-Oriented Software," Proc. ACM Symp. Foundations of Software Eng., pp. 416-419, (2011)
[13] http://in.mathworks.com/help/gads/what-is-the-genetic-algorithm.html
[14] http://www.conformiq.com/model-based-testing