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

Inclination To-wards Automated Model-Based Test Generation

Gaurav Gupta
Computer engineering,
University College of Engineering
Punjabi University, Patiala.

Parampreet Kaur
Computer engineering,
University College of Engineering
Punjabi University, Patiala

Abstract— Model-based test generation basically means functional testing for which the test specification is given as a test model. The test model is derived from the system requirements. In model-based testing, test cases are derived automatically from the test model. Coverage criteria are often considered at the test model level. The internal matters of the SUT are hidden since it uses a black-box or gray-box testing technique. Model-based tests can be applied to all levels from unit tests to system tests. Model-based testing has gained much attention with the advent of UML models in software design and development. The paper introduces model-based testing and software models as examples in addition to advantages of MBT and limitations of other approaches.

Keywords— SUT, MBT, Finite state machines (FSM), State charts, UML, Grammars, Markov chains.

I. INTRODUCTION

A model is a description of a system's behavior. Model based testing is software testing in which test cases are generated completely or partially from a model that describes some functional aspects of the System under Test (SUT). The model generates tests to cover the system behavior and when the behavior changes so do the tests. A model is a way of describing how user actions and system states in an application are related to each other [2]. If the model is depicted thoroughly describing every user action and corresponding system response, it becomes simpler and easier to automate the test case creation as well as the test execution. This technique is known as model based testing. Figure1 shows the kinds of testing, model-based testing can be applied to.

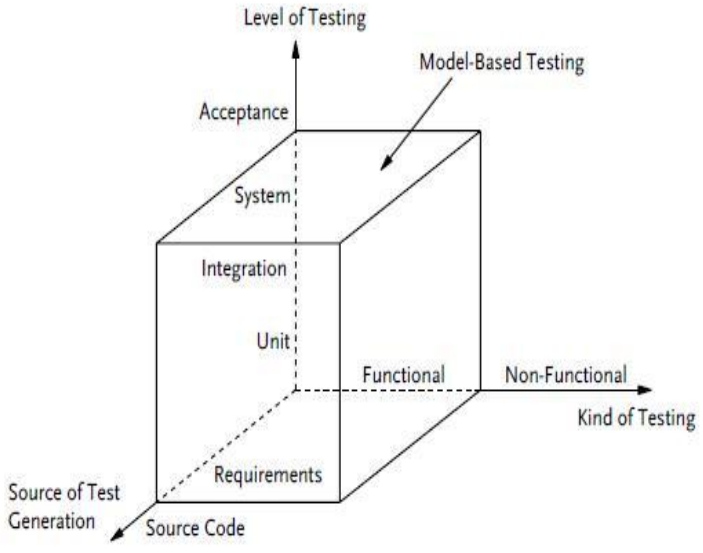

Fig 1 Fields of MBT.

Two approaches to MBT (Model Based Test Generation) are:
Offline MBT- Offline MBT means generating a finite set of tests and executing them later. It makes possible to create a tool chain. *Online MBT*- Test case generation and execution proceeds in motion. Useful for testing Non-Deterministic systems.
Thus Model-based testing typically involves the following steps:
1. Building an abstract model of the behaviour of the system under test.
2. Definition of test selection criteria. The criteria defines what test cases to generate from the model.
3. Test Oracle is a document which contains the knowledge about the expected behaviour of the SUT.

4. Generating tests abstractly from the model, using the defined test selection criteria.
5. Transforming (concretize) the abstract test cases into executable test cases.
6. Executing the test cases.
7. Assigning of a pass/fail verdict to executed test case.
8. Analyse the the execution result.

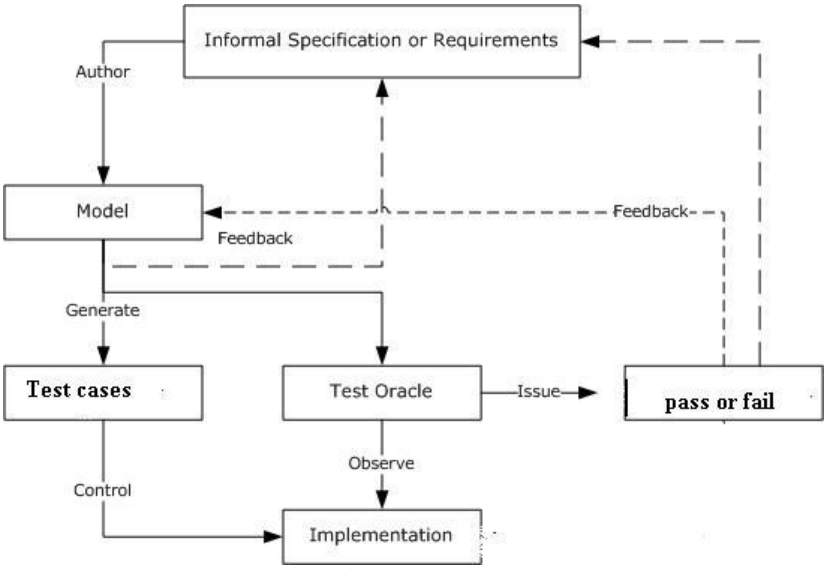

Fig 2 Working of Model Based Test Generation

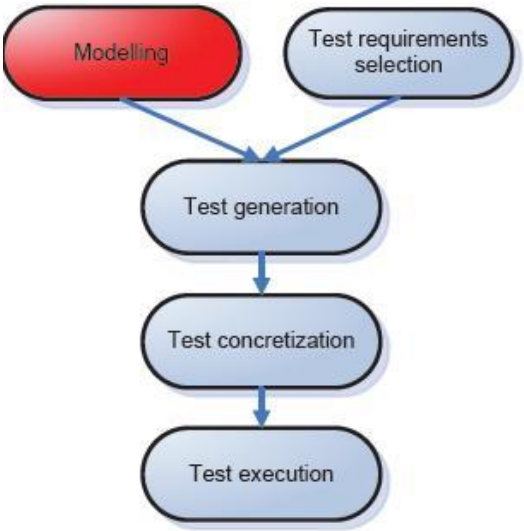

Fig 3 Steps in MBT

.

II. MODEL BASED TEST GENERATION

A model of software is a depiction of its behaviour. Behaviour can be described in terms of the input sequences accepted by the system, the actions, conditions, and output logic, or the flow of data through the application's modules and routines.

A. Models in software Testing :

There are various such models which describe different aspects of software behaviour. For example, control flow, data flow, and program dependency graphs express how the implementation behaves by representing its source code structure [3]. Decision tables and state machines, on the other hand, are used to describe external i.e. black box behaviour. In case of MBT, the testing is considered solely in terms of such black box models. The most common ways to model software are presented:

1) ***Finite State Machines :*** Finite state machine (FSM) is a behaviour model, which consists of states and transitions between those states. FSM can be described formally as quintuple (Inp, St, Tr, Fin, Lf), where

- Inp is the set of inputs to the model,
- St is the set of all states in themodel,
- Tr is a transition function which determines whether a transition occurs when an input is applied to the system in a particular state,
- Fin is the set of final states where the model can end up,
- Lf is the state where the software is launched.

2) ***Unified Modelling Language:*** The unified modelling language (UML) is quite similar to a finite state machine, except that it is used to visualize complicated behaviours of software, thus the simple graphical representation of a state machine is replaced by a structured language.
UML contains two parts: a Model, and a set of Diagrams. The Model can be described as a formal description of Diagrams, which are used for visualization , has three parts:

- *Functional model*, which shows the system functionality from the user's viewpoint
- *Object model*, which contains the system's structure such as objects, their methods and
- attributes and relationships between them
- *Dynamic model,* which shows the internal behaviour of the system using e.g. sequence, activity and state machine diagrams.

UML diagrams are divided also to three parts:

- *Structure diagrams* describe all the components which are present in modelled system such as classes, packages, objects and the overall structure of the system
- *Behaviour diagrams* are used to describe how the system actually works. This includes workflows and use cases.
- *Interaction diagrams* describe the data and control flows of the system.

3) ***Markov Chains:*** Markov chains are stochastic models, which can also be used for software modelling. These are the probabilistic state machines, i.e. the transitions of the machine contain a probability, which is used to select which transition to choose whenever leaving a state. They can be easily used to measure software's reliability and mean time to failure.

4) ***State charts:*** State charts are an extension of finite state machines. They are used to model complex or real-time systems, which are not possible easily with finite state machines. In state charts, it is possible to specify state machines in a hierarchy, where upper level states contain a complete lower level state machine. State charts are an equivalent to the Turing machine.

5) ***Grammars:*** Grammars are used to describe the syntax of programming and other input languages, and can be used to represent a model of a system in a more compact form than finite state machines do. They are easier to write and maintain.

III. WHY INCLINATION TOWARDS MODEL BASED APPROACH?

The MBT approach is associated with several boons and benefits which offer certain advantages over Manual as well as Automatic Source-Code based testing explained in the following section:

1) *Easier Test Suite Maintenance:* Both Manual and code based test generation have advantages and disadvantages [6].A problem that they share is the high effort to maintain test suites. There is no traceability from requirements to test cases. As a result, for each change of requirements, someone has to check all existing test suites manually.
2) *Ability to Detect Functional faults earlier:* Due to the missing redundant test specification, code-based testing is unable to detect functional faults. This also excludes the detection of missing functionality. But using Model-Based approach all functions can be tested.

3) *MBT allows finding Faults even before Implementation Phase:* Manual test creation is costly, and code-based test generation has the drawback that testing can only start after implementation. In contrast, model-based testing allows one to find faults before the implementation phase. The creation of formal and coherent test models results to detect inconsistencies in the requirements specification.
4) The key advantage of this technique is that the test generation can systematically derive all the combinations of tests associated with the requirements represented in the model to automate both the test design and test execution process.
5) Several studies have shown that model based testing is effective when used to test small applications, embedded systems, graphical user interfaces and state-rich systems with reasonably complex data[1].
6) Model authoring is independent of implementation and actual testing so that these activities can be carried out by different members of a team concurrently.
7) *Comprehensive tests:* If the model is a complete abstraction of the software, it is possible to automatically create test cases which cover every possible transition of the model by using graph algorithms. Model of software can help refining unclear and poorly defined requirements [7].By eliminating model defects before the coding begins and automating the test case creation the result is significant cost savings and higher quality code.
8) *Design is Spontaneous:* When a new feature is added, a new action is added to the state machine to run in combination with existing actions. A simple change can automatically move through entire suite of test cases.
Figure 4 shows the differences between current defect discovery and elimination process and early defect discovery.

IV. CONCLUSIONS

The MBT calls for explicit definition of the model, either in advance or throughout the testing process via minor changes. However, software testers of today have a difficult time planning such a modelling effort. They are victims of the ad hoc

nature of the development process where requirements change drastically .Today, the scene seems to be changing. Modelling in general seems to be gaining preference particularly in domains where quality is essential and less-than-adequate software is not acceptable. The comparison made between model based testing and traditional manual testing showed clearly that the former is the cheaper and faster method, and almost as effective in terms of the code coverage reached as the latter one. It was approximated that on yearly basis, model based testing would save significant amount of man-hours per application. There is promising future for MBT as software becomes even more ubiquitous and quality becomes the only distinguishing factor between brands [11].

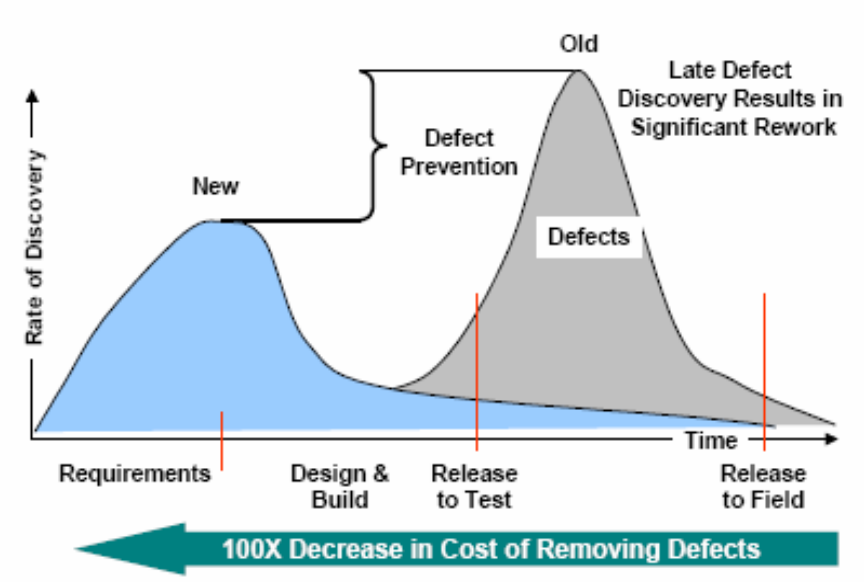

Fig 4 Savings due to Early Defect Discovery

REFERENCES

[1] Ibrahim K. El-Far and James A. Whittaker: Model-Based Software Testing.
[2] Beizer, B. *Software Testing Techniques*. 2nd Ed. New York, USA: Van Nostrand Reinhold Co, 1990. 550 p. ISBN 0-442-20672-0.
[3] Robinson, H. *Finite state model-based testing on a shoestring*.International Conference on Software Testing Analysis and Review, San Jose, California, USA, 1999.
[4] Boris Beizer. Black-Box Testing: *Techniques for Functional Testing of Software and Systems*. Wiley, May 1995.
[5] T. Quatrani. *Visual Modeling with Rational Rose and UML*. Addison Wesley Longman, 1998.
[6] Beck, Kent: *Extreme Programming Explained: Embrace Change*. Addison-Wesley,2000.
[7] Blackburn, M., Busser, R. & Nauman, A. *Why Model-Based Test Automation is Different and What You Should Know to Get Started*. International Conference on Practical Software Quality and Testing, Washington, USA, 2004.
[8] Safford, E. *Test Automation Framework, State-based and Signal Flow Examples*.
[9] Mikko Aleksi Mäkinen *Model Based Approach to Testing.*
[10] Mark Utting and Bruno Legeard *Practical ModelBased Testing a tool approach*.
[11] http://www.model-basedtesting.org

