Volume 6, Issue 3, March 2016 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Comparative Analysis of Model Based and Formal Based Software Testing Methods

Ikya Jupudy, Neha Saraf, Prof. Manjula R
SCOPE, VIT University, Vellore,
Tamil Nadu, India

Abstract— Software plays a vital role in every domain of our life, and since it is so widespread, its quality and reliability shouldn't be compromised. This is ensured by Software Testing. Software Testing is the method of evaluating the software application based on the requirements, specifications and other constraints so that there are no bugs during the execution of the software application. Software Testing finds out how much the software meets the actual requirements of the system and ensures the quality control in Software development. In this paper, the comparative study between model based testing and formal based testing has been analyzed. The paper focuses upon the problems and shortcomings of the two methods of testing. The problems faced by the testers in using model based testing or formal based testing has been discussed and the solution to it has been given. The comparison of the shortcomings also reflects which method could be more reliable in case of software testing

Keywords— MBT, FBT, Software Testing Methods, requirements, specifications

I. INTRODUCTION

Software Development is the process which involves programming, documenting, testing, finding bugs and finally developing a software application. In this process, software testing becomes a very important part since it checks whether the software meets the user requirements and specifications by verifying the application. The complexity of the software development is increasing day by day which increases the needs of testing the application. The testers are becoming the important part of the team as the software needs to be properly verified and validated.

Software Testing is done by formulating the test cases for the project based on the proper functionality of the software. From the software requirement specification (SRS), testers design the project test plan. Individual modules submitted by the developers are tested with the test cases and the requirements of the project is ensured accordingly. The results are then observed and the necessary modifications are made to yield the correctness of the product. The IEEE definition of testing is "the process of exercising or evaluating a system or system component by manual or automated means to verify that it satisfies specified requirements or to identify differences between expected and actual results."[1].

The testing effort can be divided into three parts: test case generation, test execution, and test evaluation. However, the major problem lies with test case selection. The test case is the triplet [S, I, O] where I is the data input to the system, S is the state of the system at which data is input, and O is the desired output of the system[1],[9]. Test case generation is still done manually since automatic test generation requires formal and semi-formal specifications. A test case can be understood as a structure of input and expected output. The test cases also include test case execution conditions. The problem of test case generation can be considered in two steps: firstly, test inputs must be generated and then, the next step is to provide a means of determining whether the test has passed or failed must be created.

II. WHAT IS MODEL BASED SOFTWARE TESTING METHOD?

"Model-based Testing is a testing technique where the runtime behaviour of an implementation under test is checked against predictions made by a formal specification, or model". [1]. Model based testing comes under the black-box testing approach in which the internal design of the system is not taken into account during the test case generation. The main objective of MBT is to generate functional test models based on the software requirements. Then this model is used for generating the test cases. The test execution can be done either by manual or automated test execution. The definition of MBT in Wikipedia is the following: "Model-based testing is software testing in which test cases are derived in whole or in part from a model that describes some (usually functional) aspects of the system under test (SUT)."

A model is a simplified depiction of a real system. It describes a system from a certain aspect. If a system has two different models, they are bound to appear differently since they describe the system from different perspectives. For example control flow, data flow, module dependencies and program dependency graphs [9].

We all use models to perform testing – how else will we know the status of a test, whether it passes or fails – these models give us an understanding of the behaviour of the software in a given situation[3]. The test case generation is based on the information represented by the test models, so the models obviously needs to include the behaviour that the tester wishes to test. For instance, all those test cases will be produced which incorporate the functionality whose details are provided by the model.

The attributes required by the test model is different from developers' model. The test model will contain the detail of the application of the software whereas the developers' model will contain more of a detail of the requirements and the specification [3]. When creating the test model, the degree of independence of this model from the model being used by the developers must be decided. If this high level independence is employed then the greatest benefit that can be achieved is that the test model and the development model are created at the same time.

There are two different approaches to MBT (Model Based Test Generation) are:

- Offline MBT- In this approach, a finite set of test cases are produced and executed later. A tool chain can be created using offline MBT approach.
- Online MBT- In this approach, test case generation and execution proceed simultaneously. Non-Deterministic systems can be tested using this approach [5].

III. WHAT IS FORMAL BASED SOFTWARE TESTING METHOD?

Formal Based Testing is the white box testing approach which tests the actual internal code of the program to determine the appropriate outputs. And hence, this method of testing requires the tester to have knowledge about the logic of the software. The use of formal specification or model removes the obscurity and thus reduces the chances of errors being introduced in the software development phase. This method consists of branch wise testing paths to produce predictable outputs.

Formal methods are considered as a branch of software engineering which is concerned mainly with the design of a set of development techniques and tools and their application to create better and efficient software systems. Formal methods and testing are two approaches, which are useful in developing high quality software. Formal methods have anciently been used for specification and development phases of software. However, since recently they have been used for the testing stage as well leading potential benefits.

Formal methods basically use specification languages which are based on mathematical logic. Their purpose is to aid the development process of systems and software. They are used both to describe a system and also then to analyse its functionality, verifying key characteristics of interest. It is said that the presence of a formal specification is advantageous in the determination of test cases for the software-based products. Formal methods consist of writing formal descriptions, these descriptions then undergo analysis and in some cases new descriptions are produced. They can be applied in different phases of development process.

Formal methods can be classified using two ways. Firstly, according to formal specification styles, and secondly, according to software development life-cycle perspective [4]. There are many Formal Specification Styles like: Model Based Languages, Algebraic Specification, and Process Oriented. In SDLC, formal languages are used in two phases: requirements and testing. In Requirements Analysis Phase, we describe the specification, i.e. describing the system behaviour and its desired properties. In Testing phase, we refer the formal specifications to prove or disprove the correctness of a system and hence verify the system.

IV. ADVENT OF MBT & FBT

Earlier the testing was done manually. But as the intricacy and the size of the software increases, the time and hard work required to do adequate testing grows. Manual Testing is time consuming, labour-intensive and error prone. Then came the Code based testing where the source code was used to create the test cases. Nowadays source code is not the only reference for the selection of the test cases. Recently it has been discovered that the test selection procedure can be based on the pre-code relics such as the specification of the software, requirements by the software platform and the layout models which are formulated before the process of testing is started [1]. Thus the testing skills can be exercised throughout the reinforcement process.

There are mainly two major disadvantages of code based testing. First, it is extremely difficult to mechanize and this testing depends a lot on manual test case designs. Second, there are certain forms of behaviour which are very laborious and difficult to select from the code but can be readily retrieved from layout models. For example to explain this concept, in a state diagram, we observe state based behaviour, messages are exchanged and transferred among classes in message paths which are basically different sequences. The messages are interchanged during software use and the sequences are explicitly available in UML sequence diagrams which are otherwise nearly impossible to extract from the internal code [9].

To meet all the challenges faced during the process of software testing, technologies like model based testing and formal based testing were formulated. Model based testing comes into play due to the increase in the necessity of the development's increasing pace to keep up with the technology. Apart from simply improving or automating current steps in the test process, model-based testing has become a revolution in technology which requires drastic changes in the processes and the tools for it [3].

The use of a higher degree of formalism in the development process has led to an increased demand in the current information systems. When an error that was introduced in the requirements phase is found during testing, software engineers must fix the incorrect requirements, check all of the repercussions through modifying design and implementation and finally retest the product. In order to build a secure and efficient software product and overcome the problem of insufficient budget (which occurs due to errors in requirement specifications), cost-effective methods are required to address the major hazards and that provide tangible evidence of trustworthiness.

This is where formal based testing comes into play. Formal methods are the solution to the above stated problems. Formal methods are a particular kind of mathematical tools which is used for the specification, development and

verification of software and hardware systems. The representation used in formal methods is called a formal specification language. The specification should be written formally since informal writing of specification has to be translated which makes the work even more tedious. The analyzation of the formal specification should be done as early as possible to avoid disparity and imperfection [4].

V. BENEFITS OF MBT & FBT

As can be understood by the name itself, Model based testing (MBT) is the generation of test cases and evaluation of results based on the different models of the software such as the design and analysis model. The traditional testing focussed more upon the details extracted from the code and the requirements specification which is highly contrasting to MBT. The test cases can be available at the earlier stages of software development cycle by generating the test case from design specifications, thereby making the test planning more effective [9].

One of the main problems in testing object-oriented programs is test case selection. Since systems can be simply represented by models, it is easily tractable for use in automated test case generation. On the basis of models, software development and different test activities are automated resulting in significant reductions in fault removal, development time and the overall cost overheads [9]. Some important MBT advantages can be summarized in the following points. Higher percentage of different tests coverage can be attained via this testing which is more valid for certain behavioural aspects difficult to be determined by the code. One of the major benefit of MBT is that the test cases generated from the model remains intact even if there is a change in the code due to some error.

The functional faults and the missing functionality cannot be detected by the code based testing due to missing redundant test specification. But all these functionalities can be very well detected by model based testing. There is a major drawback of code based testing that it starts after the implementation thus stressing all the responsibilities on the code itself. Moreover, manual test creation is also costly. In contrast, faults in the layout or the development process can be identified before the implementation phase in model based testing. The inabilities in the requirements specification can be detected by the creation of formal and coherent test models. The testing of small applications, embedded systems, graphical user interfaces and state-rich systems with reasonably complex data is much easier by the method of model based testing [5].

Formal specification and models helps in the reduction of errors which are introduced during the process of software development and hence eliminates vagueness. The biggest hurdle faced in this is the choice of formal specification language according to the actual customer requirements which is more complicated by the change of requirements during the development phase [2]. The possibility of the formal and automatic analysis of the relationship between the software specification and the source code can be introduced by using formal specification.

There is a possibility of creation of specifications describing the true requirements of the user by using formal methods. These specifications are not identical to the stated requirements [4]. The reason for this benefit of formal methods is the unambiguous nature of formal specifications and the ability to show certain properties about it. The implementation of a particular software as well as hardware product should satisfy the requirements specification which is ensured by formal methods. The system is required to satisfy the demand of security, reliability and correctness. This is ensured by formal methods since it acts as evidence to the satisfaction.

The formal methods is responsible for providing a measure of the correctness of system which is opposed to the current process quality. Formal Methods can also be used to detect and eliminate early design defects due to its application in the earliest design artifacts. All the possible execution paths throughout the system are considered by formal analysis tools such as model checkers. The model checker finds all the probable faults and errors. In a multithreaded system the exploration of all possible interleaving and event orderings can be done easily by formal methods, thereby solving the concurrency issue. Thus only through testing this level of coverage is impossible to achieve.

The formal description of a particular requirement helps the writer to answer to all sorts of questions in the earlier stages only thus eliminating more pressure on the coding phase. This helps to reduce the errors that occur during or after coding. Formal methods covers all aspects of the system, thus acting as a tool for completeness. A formal specification can be described abstract and precise in its and in some senses complete. A human reader can easily understand the actual picture of the software product by the help of the abstraction. There is also an added advantage of carrying out extreme level of analysis by the formality of the description. Important properties such as high level requirements or correctness of a proposed design can be easily determined by formal descriptions.

Formal methods provide the kind of evidence that is needed in heavily regulated industries such as aviation. Formal methods establish and gives firm reasons for developing the bond of trust in the product. Direct derivation of effective test cases from the required specification is easily possible by this method thus making it cost effective.

VI. LIMITATIONS OF MBT AND FBT

MBT does have certain restrictions and limitations. Needless to say, as with several other approaches, to reap the most benefit from MBT, substantial investment needs to be made. Skills, time, and other resources need to be allocated for making preparations, overcoming common difficulties, and working around the major drawbacks. Therefore, before embarking on a MBT endeavour, this overhead needs to be weighed against potential rewards in order to determine whether a model-based technique is sensible to the task at hand [9].

Not every tester can perform MBT, they need to have particular skills like familiarity with the supporting mathematics and the theories associated with the model. For example, a tester is required to have a working knowledge of the various forms of finite state machines and a basic familiarity with formal languages, automata theory, graph theory

and elementary statistics in reference to finite state models. In order to save resources at various stages of the testing process, initial efforts in planning MBT is encouraged. Right from selecting the type of model, distributing system functionality into multiple parts of a model, and at last integrating the model are all tasks involving intense labour that can forbid the outcome of such magnanimous efforts put in without a proper combination of careful planning, good tools, and expert support. The most eminent problem for state models (and most other similar models) is state space explosion.

The generated test cases may in many cases get irrelevant due to the disparity between a model and its corresponding code. MBT can never replace code based testing, since during the development process, the models generated don't necessarily contain all the details of implementation essential for test case generation. Theoretically, the models used can represent any sort of presumptions, but the modelling of some non-functional requirements, such as usability, is currently not very well perceived for model-based testing to support this area of testing [3].

Generally, actual user requirements in reality are not constant and will usually vary with time, they might be different from what the user originally states. Using formal methods don't guarantee completeness and correctness of a specification with respect to the user's informal requirements. It is very difficult to identify whether or not a given program satisfies the given specifications. For example, when using one of the verification checking approach such as Hoare logic, one needs to identify the loop invariants, which is not possible automatically. As a result, it is often not possible to prove the correctness of an existing program that has not been written taking correctness proof in account. Correctness proofs are only feasible if programming and proof go simultaneously [4].

Formal definitions of semantics of most of the important language constructs and software system components are either not available or too complex to be useful. A formal description of the program should contain a description that a program is to work in coordination with hardware and under the specification of operating system in order to prove the correctness of the program. Generally for the technical environment in industrial software development, such a formal description is often not available. A worsening limitation is that such a formal description has to take a very distinct form depending on the particular formal method used.

.

VII. CHALLENGES AND SOLUTIONS

In future, it is very important to work on the fitting of specific models (finite state machines, grammars or language-based models) to specific application domains [1], [9]. For working on these type of models it is necessary to invent new models because mentals models will be transformed into actual models. There will be special purpose models and general models. The former will be made to satisfy very specific testing requirements and the latter will be made from any number of pre-built special-purpose models.

- Finding suitable abstractions is difficult
- We cannot execute partial tests

There is a need of alternative styles of testing for recognising large number of problems that prevents productivity of the software. We must form an understanding of how we are testing so that it is easy to communicate the understanding. This will help to create a model for the software process which can be further encapsulated with the testing insights and help everyone to benefit from it. These goals can only be achieved if the models evolve from mental understanding to artifacts formatted to achieve readability and reusability.

Formal methods are descriptive and analytical in nature. They are not considered to be creative. In reality, there are only formal ways of describing and analysing designs. There is no such thing as a formal design process. Formal methods should be combined with other techniques to develop a real system. Software product quality is not at all affected by the use of formal methods. Therefore, a lot of hard work is required by the successful providers of software so that all the important aspects of a software product is addressed by them. Normally all the inputs are taken from external environments by the software system. These inputs may not be predictable. This creates a lot of problem. A lot of difficulty is faced in developing 'correct' specifications and deciding the correct behaviour. There is not help or contribution from formal methods towards this aspect of software system.

VIII. CONCLUSIONS

This paper presented the comparison between Model based and Formal based software testing methods. With the trend of developments in each of these methods, we certainly cannot state one to be better over the other, as each method has its own set of benefits and drawbacks. To conclude, the usage of MBT reveals substantial benefit in terms of increase productivity and reduced development time and costs but MBT can't replace FBT since models are abstract higher level representations and lack of several details which are present only in the code.

REFERENCES

[1] Hitesh Kumar Sharma, Sanjeev Kumar Singh and Prashant Talawat, *Model Based Testing: The New Revolution in Software Testing*, Database Systems Journal, Volume 5,no.1,2014

[2] Monika Singh, *Formal methods: A Complimentary Support for Testing*, International Journal of Advanced Research in Computer Science and Software Engineering, Volume 3, Issue 2, February 2013.

[3] Stuart Reid, *Model Based Testing*

[4] Mona Batra, Amit Malik and Dr. Meenu Dave, *Formal Methods: Benefits, Challenges and Future Direction*, Journal of Global Research in Computer Science, Volume 4, No. 5, May 2013.

[5] Gaurav Gupta and Parampreet Kaur, *Inclination towards automated Model based test generation,* International Journal of Advanced Research in Computer Science and Software Engineering, Volume 3,Issue 3,March 2013

[6] Sunil Kumar and Guide Dr. P.K Yadav, *Formal Methods of Software Testing and Terminology,* Int. J. Comp. Tech. Appl., Vol 2(4), July-August 2011.

[7] Sanjeev Dhawan, Nirmal Kumar and Shiva Saini, *Model Based Testing Considering Steps, Levels, Tools & Standards of Software Quality,* Journal of Global Research in Computer Science, Volume 2, No. 5, May 2011.

[8] Eckard Bringmann and Andreas Kraemer, *Model-based Testing of Automotive Systems,* 2008 International Conference on Software Testing, Verification and Validation.

[9] Santosh Kumar, Subhendu Kumar Pani and Durga Prasad Mohapatra, *Model Based Object Oriented Software Testing,*Journal of Theorotical and Applied Information Technology.

[10] Marie-Claude Gaudel, *Formal Methods and Testing: Hypotheses, and Correctness Approximations,* FM'05 Proceedings of the 2005 International Conference on Formal Methods, pages 2-8, 2005.

[11] Mike Barnett, Wolfgang Grieskamp, Lev Nachmanson, Wolfram Schulte, Nikolai Tillmann and Margus Veanes, *Model-Based Testing with AsmL.NET,* 1st European Conference on Model-Driven Software Engineering, pages 12-19, 2003.

[12] Rasneet Kaur Chauhan and Iqbal Singh, *Latest Research and Development on Software Testing Techniques and Tools,* International Journal of Current Engineering and Technology, Vol.4, No.4(Aug 2014).

[13] Monalisa Sarma, P.V.R. Murthy, Sylvia Jell and Andreas Ulrich, *Model-Based Testing in Industry- A Case Study with Two MBT Tools,* AST'10 Proceedings of the 5th Workshop on Automation of Software Test, pages 87-90, 2010.

[14] Jonathan P. Bowen, Kirill Bogdanov, John A. Clark, Mark Harman, Robert M. Hierons and Paul Krause, *FORTEST: Formal Methods and Testing,* Computer Software and Applications Conference, 2002. COMPSAC 2002. Proceedings.26th Annual International, pages 91-100,26th-29th Aug, 2002.

[15] Jan Tretmans, *Testing Concurrent Systems: A Formal Approach,* CONCUR '99 Proceedings of the 10th International Conference on Concurrency Theory, pages 46-65, 1999.

[16] Mark Utting, *Position Paper: Model-Based Testing,* Verified Software: Theories, Tools, Experiments (VSTTE) Conference on the "Program Verifier".

