International Journal of Advanced Research in Computer Science & Technology (IJARCST 2015) Vol. 3, Issue 4 (Oct. - Dec. 2015) ISSN : 2347 - 8446 (Online) ISSN : 2347 - 9817 (Print)

Software Testing With NFR Using Formal Specification and Model Based Technique - A Step Ahead to Produce High Quality Software

[I]L. Amudha, [II]T.M. Nithya, [III]I. Infant Raj
[I,II,III]Assistant Professor, Dept. of CSE, K. Ramakrishnan College of Engineering

Abstract
This paper provides an overview of model based testing using formal language specification. Testing is an inevitable module in software development that ensures the correctness of a software. Although many methods and tools are available for testing no software is 100% complaint free. A good testing should aim at finding more bugs rather than hiding them. Our current work is to find a better methodology to improve the quality of software. Model based testing can be sued to represent the behaviour of the system and used of formal models will reduce the development and maintenance cost, thereby also improving the quality of the software developed. We propose a combined approach of two difficult and rare techniques i)Non-functional requirement testing and ii)Formal specification. Test cases generated with this method proves the improvement in finding bugs in the software.

Keywords
Non Functional Requirements, Model Based Testing, Formal Languages, State Diagrams, Test Cases, Test Generation.

I. Introduction

We use testing to find whether our software produces the expected results or not. Testing is now being automated by software tools that generated test cases. But many of today's automatic test generating softwares use black box and white box testing and deterministic finite automata. But a model more powerful is required to describe the operational behaviour of the software and find defects and shortcomings to a huge percentile. The following sections describe i) why functional requirements which are preferred by many and what are its limitations, ii) Why NFR is not widely preferred by many, its limitations and benefits, iii) How formal specifications can help to build a model that is more accurate in finding defects.

Model based testing (MBT) is an application of model based design for designing and optionally also executing artifacts to perform software testing or system testing. Models can be used to represent the desired behaviour of a system under test (SUT), or to represent testing strategies and a cost environment. Several model based languages are available such VDM, Z, B is available to build a model of the intended behaviour and languages.

NFR is considered to produce high quality acceptable software. Most of software developers and testing people prefer functional requirements for its simplicity. Comparison of Functional requirement testing and non-Functional requirement testing is given in Table 1.

Mostly Functional testing follows white box testing methods with fixed steps whereas Non functional testing uses behavioural or black box testing strategies[1].

Table 1 : Comparison of FR and NFR testing methods

FR Testing	NFR Testing
'What' the system must do?	'How' the system must accomplish 'What'?
Involves product features and functionality	Involves quality factor
Test is done through simple steps to check with expected results	Testing yields huge data set. Analysis is needed
Focus : Defect Detection	Focus: Qualification of results
Testing involves unit & integration level	Testing is system level
Failure is normally due to code	Failure is due to code design and architecture.
Testing is easy because of well defined goal	Difficult due to its inability to operationalize the soft goals into concrete testable objective
Clear pass/fail criteria	Results must be quantitatively documented

II. Process of Model Based Testing

Further to the above points unlike functional testing, non-functional testing requires configuration changes for each test case. Also NFR testing requires knowledge and experience of product domain, design, architecture and statistical skills.

1. Advantages of Using Model Based Language

By using testing and formal models together, the software development cost can be reduced, by applying the testing steps well ahead, before they become a big problem at the end, if it is not identified and eliminated in the early stages of development. MBT models real life situations more concretely and yields good set of test cases from abstract formal models.

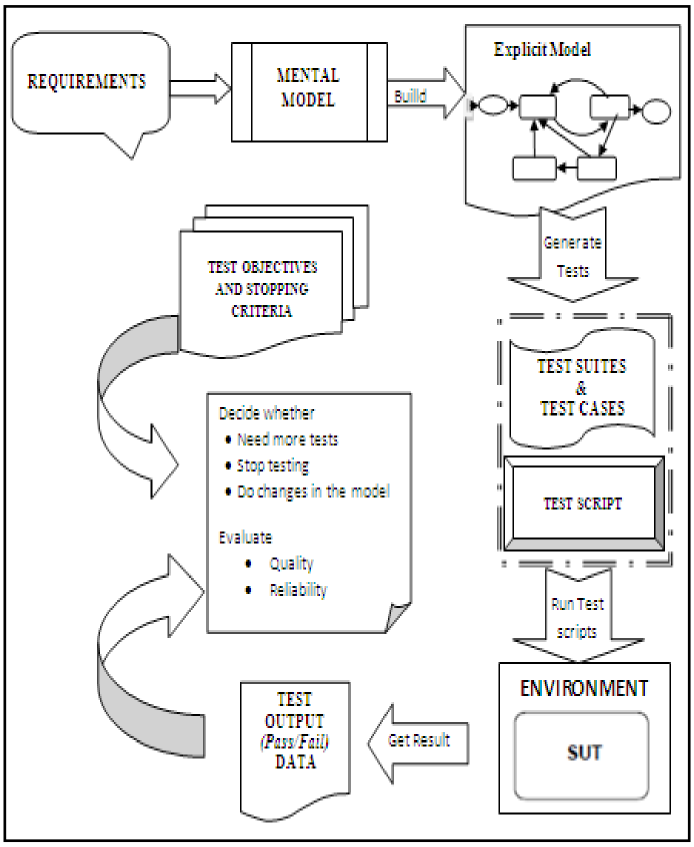

Fig. 1 : Process flow of MBT

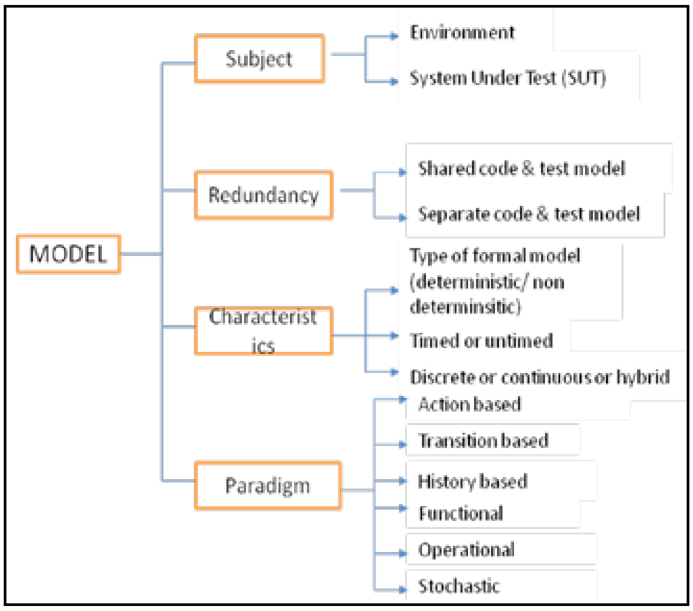

Fig. 2 : Types of model based testing

III. Formal Methods

The formal specifications makes it more easier for the person who is testing the software, to be more clear about what is expected for a system to test? There are many formal specifications like Finite state based language, Specification language, Model based languages and Algebraic languages[3]. Out of which our concern is about Finite state based language and its methodologies.

2. Model Based Paradigm

Utting et al has discussed about seven dimension of model based testing[5].
The subject of the model can be either the intended behaviour or the environment in which the software is going to be used.
The second classification defines the redundancy, where 2 methods can be adopted: one where code generation and test case generation uses a common method (shared model) and the second method uses different methods for code generation and test case generation. (separate test model).
The third classification is about the characteristics of the model: it may be deterministic or non-deterministic, with changing timing constraints and discrete, continuous or hybrid systems may be under test.
The fourth dimension is the model paradigm. Many notations exists to describe the model :

i) State-based,
ii) Transition-based,
iii) History-based
iv) Functional
v) Operational,
vi) Stochastic and
vii) Data-flow.

These different modelling notations can be used for behavioural testing models.

1. Finite state based language

At any state a tester has a specific set of inputs to choose from. This set of inputs varies depending on the exact “state” of the software. This characteristic of software makes state-based models a logical fit for software testing: software is always in a specific state and the current state of the application governs what set of inputs testers can select from.
Finite state machines/finite automata have been around even before the inception of software engineering. Using finite state models in the design and testing of computer hardware components has been long established and is considered a standard practice today[4]. When Finite state machines are used for software design and testing a lot of coding time is saved and the quality of the software developed is improved.
A finite state machine can only be in one state at any one time. The occurrence of a transition from one state to another is exclusively dependent on an input in *I*.
A finite state machine can be a state diagram or a step by step process diagram that reaches a destination after some finite set of actions or input sequence.

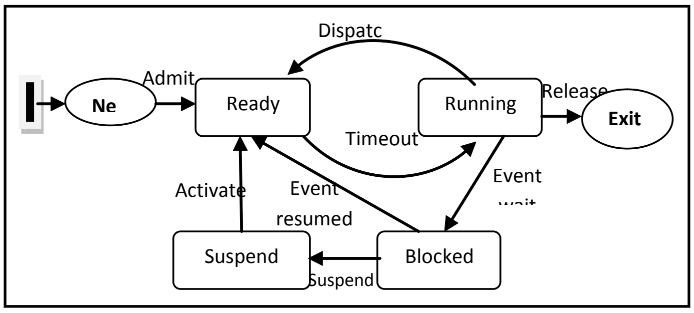

Fig. 3 : State diagram of any process / product

Formal definition of finite state machines:
A finite state machine representing a software system is defined as a *quintuple (Q,I, T,F, D), where*

- Q is the set of all states of the system.
- I is the set of inputs of the system
- T is a function that determines whether a transition occurs when an input is applied to the system in a particular state.
- F is the set of final states the system can end up in when it terminates.
- D is the destination state into which the software is launched.

2. Model Based Formal Specifications Testing Methods

A. Z-Notation
Z is based on the standard mathematical notation used in axiomatic set theory, lambda calculus, and first-order predicate logic. All expressions in Z notation are typed, thereby avoiding some of the paradoxes of naive set theory. Z contains a standardized *mathematical toolkit* of commonly used mathematical functions and predicates.

B. B Method
The B-Method is a formal method that uses concepts of First Order Logic, Set Theory and Integer Arithmetic to specify Abstract State Machines. Beginning with an informal set of requirements, which is usually written using natural language, an abstract model is created[7]. The B-Method's initial abstract model is called Machine. A machine can be refined into one or more Refinement modules. It is derived from a Machine or another Refinement and the conformance between the two modules must be proved.

C. VDM
Vienna Development Method Specification Language covers the interface-requirements of a web-based application. It uses a framework to support the transformation of the conventional SRS to a design specification, and a Finite State Machine based verification model, to test the design specification against the SRS[8].

IV. Generating Tests
Generating tests from a model depends on the nature of the model. Finite state machines, is as simple as implementing an algorithm that randomly traverses the state transition diagram and moves through the sequences of arc along the generated paths. They define the test and outcome. The sequence of labels in Fig 3. decides whether the process ends after a finite number of steps or not.

V. Analysing Test Results
Evaluating test results is perhaps the most difficult step in testing process. Testers must determine whether the software generated the correct output given the sequence of test inputs applied to it. In practice, this means verifying screen output, verifying the values of internally stored data and establishing that time and space requirements were met.
During traditional testing, tests are conceived and executed one-at-a-time or in batch. Suppose, e.g., that a tester was able to run 300 test cases and exactly 50 bugs were found as a result. At the end of such a test, one can claim only that 300 tests were run and 50 bugs were found. What if the 301st test case makes a wide change in the results summarized. MBT does verification of states at this point.
Studies show that testing a variety of applications has been met with success when MBT was employed.

VI. Limitations in Model Based Testing
Almost every research on model based testing agrees on one thing: deployment of model based testing into an organization requires considerable efforts and investments.

1. Testers should be extremely skilled.
The people who do testing should be more familiar with the software model and should know the concept of state machines with their variations, input and output parameters. They should be skilled in Automata theory and languages and tools used for testing.

2. A large initial effort in terms of man-hours is required.
Since all constraints must be considered the software need to be modularized and more people should be put to work and takes more hours for launching the tests and integrating together again.

3. Models themselves have also several drawbacks
Even a very small software application may have too many states to comply with all tasks and requirements. In such case the maintenance becomes a tedious task.
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.

VII. Application Area
Before using model-based testing, we must be sure that the approach is suitable for the current environment and application. The creation of test models is clearly very large but this must be recouped by the lower maintenance costs when the system is ready for use. Low maintenance costs may be predicted, for example, if the system is planned to have only a short operational life or it is expected that there will be mainly few changes to the system required by the users (and its environment). Obviously the application must be appropriate for modelling in a supported notation. Many switching applications have been found to be mainly suitable, as they are well-suited to modelling as a state model and there is good tool support for switching applications [4]. The application should also be considered significant enough to warrant the cost of model-based testing. If high quality is not important to the customer then model based testing will be of less use in terms of cost efficiency for the users

VIII. Conclusion
Testing based on formal specification guides test case selection by precisely pointing out the behaviour that needs to be tested. The main goal is to cover all possible behaviours that can be observed from specification. This current paper focuses on formal specifications in deriving test cases using MBT. The results can be even further improved, if this method is combined with unifying theories of programming in finding the defects in a software [6, 11, 12].

References

[1] *International Journal of Software Engineering & Applications (IJSEA), Vol.3, No.4, July 2012 ISSUES IN TESTING OF SOFTWARE WITH NFR,Pratima Singh and Anil Kumar Tripathi.*

[2] *Practical Model-Based Testing: A Tools Approach, Mark Utting and Bruno Legeard, ISBN 978-0-12-372501-1, Morgan-Kaufmann 2007.*

[3] *International Journal of Advanced Research in Computer Science and Software Engineering Formal methods: A Complementary Support for Testing, Monika Singh*

[4] *Using Formal Specifications to Support Model Based Testing ASDSpec: A Tool Combining the Best of Two Techniques, A.P. van der Meer, R. Kherrazi, M. Hamilton*

[5] *A Taxonomy of Model-Based testing, Utting, Pretschner and Legeard.*

[6] *Testing techniques in software engineering, Paulo Borba, Ana Cavalcanti, Augusto Sampaio, Jim woodlock, Eds, Springer Edition, ISBN 978-81-322-1478-7*

[7] *Complementing the B-Method with Model-Based Testing? Ernesto C. B. de Matos Federal Univeristy of Rio Grande do Norte*

[8] *A VDM-based Approach for Specifying and Testing Requirements of Web-Applications Souvik Sengupta, Ranjan Dasgupta, International Conference on Information and Communication Technologies (ICICT 2014)*

[9] *E. C. B. Matos and A. M. Moreira. BETA: A B Based Testing Approach. In R. Gheyi and D. Naumann, editors, Formal Methods: Foundations and Applications, volume 7498 of Lecture Notes in Computer Science, pages 51-66. Springer Berlin Heidelberg, 2012.*

[10] *Müller A. VDM—The Vienna Development Method. Bachelor thesis in" Formal Methods in Software Engineering", Research Institute for Symbolic Computation (RISC), Johannes Kepler University Linz, Austria;2009.*

[11] *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 41, NO. 10, OCTOBER 201,"5Automated Checking of Conformance toRequirements Templates Using NaturalLanguage Processing" Chetan Arora, Mehrdad Sabetzadeh, Member, IEEE, Lionel Briand, Fellow, IEEE, andFrank Zimmer, Member, IEE.*

[12] *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 41, NO. 11, NOVEMBER 2015,"A Survey on Load Testing of Large-Scale Software Systems", Zhen Ming Jiang, Member, IEEE and Ahmed E. Hassan, Member, IEEE*

[13] *Software Engineering: A Practitioner's Approach, Roger S.Pressman, McGraw-Hill Higher Education;*