Model Based Testing

Manoj Philip Mathen
Independent Validation Services
Infosys Technologies Ltd
Mysore, India

Abstract-- This paper is a quick glance into what is Model Based Testing, its evolution, its current state, who should use it, the techniques and tools involved and what will it mean to the enterprise in terms of the cost incurred. The paper starts with a brief exploration into some of the building blocks of MBT, followed by an attempt to define Model Based Testing. This is followed by an argument on the need of Model Based Testing, and a quick overview on Model Based Testing and the approach. Next, we walkthrough 2 example scenarios, where certain models have been exhibited to showcase testing benefits. This also shows the different techniques and methods in MBT. Finally the author describes some common challenges in Model based Testing and some best practices.

Index Terms-- Automating models, Automated test design, BGM, Early Defect Detection, Finite State, MBT, Models, Switch, Understanding Systems, UML

INTRODUCTION

Model Based Testing (MBT) is very common in validating embedded systems, phones, switches, etc. The model based testing was very successful and has yielded good results in these areas. This could have been the reason why practitioners tried exploring the feasibility of the same in other areas of Software validation. There are many research studies that are ongoing and many QA schools have embarked on the MBT journey. The results from those who are travelling the MBT journey are encouraging, but not conclusive. There has to be more exhaustive research / analysis.

Teaching addition / subtraction using models (i.e. fruits, sweets, abacus counters, etc.) makes it very easy for the end-user (student) to understand the concept and the system itself. This might also answer the common questions kids ask, if 2+2 = 4, how come 3+1 is also 4!

The results indicate that the model based approach increases the effectiveness of learning as well as reduces the cost of learning. Does this hold true in Software Testing?

MODEL BASED TESTING

System Models are very common in the aviation and telecom industries. When Boeing designs a new aircraft such as the 787 Dream line or the 747-8, they model the aircraft in software and it is run in simulated runways and skies to understand the aircraft behavior in varying environmental conditions. Models thus explain what otherwise we don’t understand fully. It abstracts and simplifies things. It helps us answer what-if questions. The very task of modeling forces the designer / modeler to break down the complex issues into easily assimilate-able form.

Model based Testing is thus a natural way to approach any system/concept or solution. Models can be as simple as a graph or a flow chart. Complex models can be created using visualization tools or pseudo code. The use of such ‘models’ in validating a system is usually called as model based testing. A model alone will not suffice the testing requirements, hence the existence of ‘test models’ that can generate test cases. MBT takes ‘modeling’ to the next level. Models are used not just to simplify/abstract systems, but also to auto generate use/test cases.

A. Models : The Building Blocks

1. Finite State Machines.

A Finite State machine (FSM) is a mathematical abstraction used to design digital logic, business workflow or computational programs. It is a behavioral model composed of states, events and transitions. Finite State machines are used widely in problem solving, artificial intelligence, circuit designs, etc. The below is a FSM depiction of an electric switch. The switch has only
2 different states - ‘ON’ and ‘OFF’. There are 2 Events – ‘Switch On’ and ‘Switch Off’ and the process of ‘state change’ is known as transition. The Switch is currently in ON state. An event ‘Switch off’ can change the state to OFF state.

Figure 2: FSM Depiction

The Finite State Machines thus describes a collection of states and their associated transitions.

2. Finite State Diagrams

The finite state machines can also be represented using state diagrams. A state diagram is a tabular representation of the machine. The above State machine can be depicted as below:

<table>
<thead>
<tr>
<th>Event</th>
<th>Current State →</th>
<th>ON</th>
<th>OFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Switch ON</td>
<td>..</td>
<td>ON</td>
<td></td>
</tr>
<tr>
<td>Switch OFF</td>
<td>OFF</td>
<td>..</td>
<td></td>
</tr>
</tbody>
</table>

The above ‘switch’ example is a very simple one, depicting only 2 states. The switch has to be either in a switched on state or switched off state. The above table shows that if the Current State is ‘ON’ and if the ‘switch on’ activity is performed, there will be no change in the state. If the Current State in ‘ON’ and if the ‘switch off’ activity is performed, the state will be changed to ‘OFF’.

The FSM and the state diagrams will be used throughout this paper to explain the Model based Testing concepts. Creating a model is thus not so difficult, more difficult is the question what to model and what not to model.

Example Scenario

Consider a simple web based sales application. This has a home page, which has content about the sales application and other details. It has 3 links (L1, L2 and L3) which takes us to the Inventory page, Sales page and the Payment Page. Figure 4 shows the State diagram for our sales application.

From any page (except Payment.html), the user can go to the home page (using L8, L9). A user on the payment page cannot come to the Inventory page directly. Below depicted is the state diagram for the same and we see there are 4 states and about 9 transitions. The state diagram helps us understand the web based system more easily. It also helps us in asking questions around new / possible transitions.

Negative test scenarios can also be tested using the State Diagram. For example, the below state diagram depicts that a user on the Payment page cannot go back to the home page. This design might be due to security concerns. Similarly, a user cannot go from the Payment page to the Inventory page directly.

Figure 4: State Diagram

B. Model Generation

Building a model is time consuming. But this is a one time activity and experience makes it easier. Building a model helps you in exploring the application in more depth. It makes you think “What-if” at all states. This helps you in finding out all possible use-cases, which in turn will help you find gaps in the requirement at a very early stage itself. The cost-benefit of finding a defect in the requirements phase it is very evident.

1. Automating Models

Automation for models focuses on automating the transitions and verifying the states. The business intelligence behind state transitions will be inbuilt into the model and the state transformations need to be verified in an automated manner. The automated validation checks
should also validate that at any point, the model is in an acceptable / predefined state. Models can be automated in different technology. A UML based model is easy to decipher and built-upon. A complete automated model will, do the following.

- Mimic the system behavior based on varied inputs.
- Validate the output.

A big assumption is that the model is a very similar replica of the system. To facilitate this, the model generator should have the entire/abstracted business logic/use cases built into it. This in short, is the model creation effort. A subject matter expert involvement is recommended at this state. If the model is not capable of validating the outputs or generating testcases, then it is just a model, and may not be of much use for testing. However such models are still a very powerful tool for the design/development team.

2. Bayesian Graphical Modeling (BGM)

This is a modeling technique where a graph denotes the conditional dependency between various factors. They are commonly used in statistics and machine learning. If the network structure of the model is a directed, non-cyclic graph, the model represents a factorization of the joint probability of all various factors.

e.g.: if the Events are X1, X2, Xn, then the joint probability is

\[
P[X_1, X_2, \ldots, X_n] = \prod_{i=1}^{n} P[X_i | p_{Xi}]\]

Let us take an example.

The Build Quality assessment of a particular deliverable

Before we start testing, we can guess how much of an effort will be required – factoring the build quality. Now this depends on multiple factors like Code Complexity, Testing accuracy, Defects detected during design review, etc.

![Figure 5: BGM](image)

2. **Benefits of Model Based Testing (MBT)**

Models are gaining importance in Software Development. It helps in understanding requirements, validating requirements, building designs and guiding other SDLC activities. Modeling languages like UML come handy in modeling software. The benefits realized from Model based Testing are numerous. The increased focus on the same is a testimony. The below table illustrates some of the common benefits of Model based Testing.

<table>
<thead>
<tr>
<th>Benefits of Model Based Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Best way to Communicate within the team</td>
</tr>
<tr>
<td>2 Economical means of knowledge capture</td>
</tr>
<tr>
<td>3 High Degree of Automation feasible</td>
</tr>
<tr>
<td>4 Ability to Generate Regression Suites</td>
</tr>
<tr>
<td>5 Early Defect Detection</td>
</tr>
<tr>
<td>6 Good Reference</td>
</tr>
</tbody>
</table>

Testing can be very productive using Models. Most of the black box testers today use models unknowingly, for e.g., a tester drawing the functionality of the system that they are testing on a whiteboard, tracing the program flow, etc. If these testers use the Model based approach in a more structured and scientific manner, the benefits that will befall on them are numerous. Let us explore the benefits of Model based testing in more detail.

1. Best way to communicate within the team

Often it so happens that isolated pockets of knowledge do exist in large project teams. The development team might miss...
the big picture of the system. They will have the ‘depth’, but ‘width’ might be missing. The testing team might have the width, but it might be too shallow so, they may not be aware of some of the need-to-know details of the system. Having a model is a good way of ensuring everyone has the same bird-eye’s view. The process of model creation is also an interesting activity. A lot of learning happens in this phase.

2. Economical means of knowledge Capture
The model will be a one-point reference for any new joiners within the team. Complex system details are usually transitioned multiple times, especially when there are late entrants into a project team. Modeling has a significant 1-time effort, however in the long run the investment will be justified. The model also is a one-stop place to understand the system.

3. High Degree of Automation feasible
System behavior is predicted by the model and it also generates test cases. Executable test cases can be auto generated from these models. The benefits are mostly seen in systems having complex features. The traditional test design methods used to generate test cases became too expensive and labor intensive when the applications/system became increasingly complex.

4. Ability to Generate Regression Suites
Once the test cases are auto generated, a subset moves into the regression bed. Since a working model is available, the same can be validated against the model. Subsequent releases will introduce new changes into the model and this will have to be updated in the model.

5. Early Defect Detection
Testers are involved from the beginning, this forces testability into the design, complete test coverage and coverage of complex functionality. The cost pressure in the industry introduces a constant in the cost equation. This constant is the cost for cost reduction. Factors such as early defect detection will ensure such costs are negated.

6. Good Reference
A good model is a good reference for the system. It reflects the capabilities of a system, its core functionalities. It also is a good study-aid for anyone doing a system study or analysis. However, the model should be continuously updated to reflect any major change in the system. Model Based Testing, is capable of automating the test design itself. Normally, the test engineer designs the test cases from Requirements. The test cases thus developed can vary based on who designed/developed the test cases. Understanding of the requirements might differ amongst different people. In a model based scenario, the test engineer doesn’t jump into test cases directly. Instead, he understands the requirements first, and creates a model of the software application. In this process of ‘modeling’ the software itself, many a defects and missed requirements are caught. A defect detected as early as in the requirement phase is the least expensive defect to fix. Once a model is created, test cases can be auto generated. Some schools treat Flow charts, Data Flow Diagrams, Use Cases, State transition tables, etc. also as models.

The ability to abstract at the right level will be the key for a successful model. A model shouldn’t have too much of an information, neither too less. Creating a working model from the requirement is the most crucial step. Once the model is created, test cases can be auto generated from the same. The time taken to create the model might be lesser than the time to create the test cases manually. Modeling a system also calls for a systematic testing of the same. Modeling also helps in base lining the requirements. E.g.: What happens when you click on submit with ‘xyz’ as inputs. This may not have been even thought of.

The finite state machine / finite state diagram(s) can be used to depict the various states within a model. These states refer to the different ‘co-ordinates’ that a system reaches during operation. Different algorithms, like the Chinese Postman algorithm can be used to generate test/use cases that traverse all these states/co-ordinates.

D. Challenges in Implementing Model Based Testing (MBT)

Model based testing also involves many challenges. Lack of awareness about the system, skill set challenges, management support, and tools availability are few of them.

1. Awareness
The concept of Model based Testing itself is not very common among the tester community. Many test cycles get completed even without a thought on MBT feasibility. Trainings & Seminars will help on this front. The tester should be made
aware of this concept in the first place. The benefits and how it addresses many a pain point should be explained.

2. Complicated model - State Explosion
While modeling a system, ‘level of granularity’ is very important. It shouldn’t be fine grained or too much coarse grained. Usually, the modeler gets so much information about some area and he depicts all of them in excitement. This results in too many states and transitions in the model. This state is known as state explosion.

3. Lack of Information
Many a time, during the modeling phase, sufficient information may not exist to facilitate an optimal model. This might either delay the modeling process or result in an incorrect model. The latter is more risky. Modeling should be seen as a phase where requirement/design validation happens.

4. Stake holder buy-in
Management questions the effort/cost associated with this extra task of modeling the system. The return on this investment may not be easily visible or well-articulated. This is mainly an awareness challenge and the modeling concept should be familiar to both the developer and tester community. Many a time, management refuses to make a one time investment in creating the model. The cost incurred may not justify the need.

E. Tools used for Model based Testing

There are numerous tools today in the market which facilitates Model Based Testing. Experience reveals that a proper analysis has to be done before shortlisting a particular tool. Is the model going to auto generate test cases/scenarios, UML/XML compliance, other downstream tools that the QA team might use etc. are some factors to be considered.

- Nmodel from Microsoft
- Spec Explorer for Visual Studio
- A traditional method is to use Visio to call out all the states and the associated variables, outputs from each state etc.
- UML Pad
- MaTeLo. This tool uses markov chains for modeling the test.

CONCLUSION

Model based testing is thus a growing area. The benefits of MBT are many and it has to be socialized with the testing community. Fitment analysis of MBT is also a key. Knowing the concept of MBT shouldn’t result in incorporating MBT anywhere and everywhere. Some candidates will be core applications, legacy systems, systems with costly upstream interfaces, etc. Right model (right level of abstraction / granularity) is one of the key to success with MBT. Virtualization in modeling is a growing area and the benefits are manifold. This brings the benefits of models & virtualization together. Research and studies are in progress on this front.

ACKNOWLEDGMENT

I gratefully acknowledge the contributions of my colleagues in the Middleware Testing team at Infosys. Their valuable comments, research ideas, reviews have contributed into the current version of this document. Many a concept discussed in this paper has been tried out in the real world by them. Thanks guys!

BIOGRAPHY

Manoj Philip Mathen is a technical consultant and a Project Manager with Infosys Technologies Ltd, India. He comes with rich experience in Middleware Testing and Development. An author, speaker at various forums, he has authored multiple white papers. His domain exposure includes Transportation, Banking and Insurance. His current interests include model based testing, services interoperability & service governance.