Model Based Testing - Executable State Diagrams

Annamariale Chandran
AVACorp Technology, Chennai - 600034

Abstract- Model based testing that gained popularity in recent past describes models mostly for an application incorporating complex algorithms with the use of some automated tools. These readily available tools mostly focus on having inbuilt algorithms that will be specific to a set of application framework. Even the claimed auto generated tools require a pre-requisite of some form of inputs as models to feed into its system that will provide a set of test cases for test execution. This paper brings in an innovative approach of how to derive models that can be executed directly with less dependency on writing manual test cases. This was also proved for high reusability and maintainability. As the knowledge on model generation is must either with or without the use of automated tools, the insight on how to manually generate the models, its parameter considerations, different techniques to generate system model for test execution along with the advantages and further scope of enhancements will be detailed in this paper.

Index Terms-- Model based testing, State Diagrams, Test Model Generations

INTRODUCTION

Model based testing (MBT) is a break-through innovation in software testing as it ensures a repeatable, maintainable run for execution supported by scientific basis for good coverage for all the behaviours of the application under test. Testers using this model concentrate on a use case or data model generation instead of hand crafting individual test cases. More over the generated models provides a confidence on the system understanding, starting pole, integration scenarios and avoids duplication of test conditions. It also supports to draft low level test cases in the order of test execution. The generated model also offer high readability and fastens review effort. The approach of model based testing involves developing models initially to draft the final test cases for execution. The model essentially reflects functional specification and to some extent, its application design inputs as well. Model generations are also governed by specific industry standards that can be further enhanced by experience standards.

While model based testing alone is not a solution, it offers considerable promise in reducing the cost of test generation that increases the effectiveness of the tests and shortens the test life cycle. Moreover, model based testing can be proved to be effective for agile life cycle models because the models can be easily traced for changes in requirements so as to update models and then to rapidly regenerate the test suites. This will avoid tedious and error prone editing of a suite of conventional test cases.

MODEL BASED TESTING IN PRACTICE

Model based testing is a derivative of Unified Modeling Language (UML) concepts and this section will detail on how UML concepts are used in model based testing that in turn is used in pre-system and in system testing engagements. UML is a visual language for specifying, constructing, and documenting the artifacts of systems. Complex software designs that are difficult to describe textually can be readily conveyed through diagrams using UML. As such UML is not an industry standard but offers mechanisms for customization.

Broadly, UML diagrams are classified as structure diagrams, interaction diagrams and behavior diagrams. Structure diagrams are used extensively in documenting the architecture of the systems, interaction diagrams are used to model the communication protocols and behavior diagrams are used extensively to describe the functionality of the systems. As behavior diagram is mapped with system testing requirement, this again is broadly categorized as per below representation.

Activity Diagram
Behavior Diagram
Use Case Diagram
State Machine Diagram

Fig1: UML representation for Behavior Diagram

Activity diagram represents the business and operational step by step workflows of components in a system. Use Case diagrams mostly represents the functionality provided by the system in terms of actors, their goals represented as use cases, and any dependency among the derived use cases. But state machine diagrams represent the states of the system as the events occur. As first two representations are mainly intended for requirements phase, the state machine diagram is mostly found to be relevant to use in testing phase. This is because of the fact that state diagrams are mostly aligned with the concept of testing that is mainly concerned with how the behavior of the system changes on subjecting it to some
inputs. State machine diagrams are further supported by finite state machines and state transition tables. The brief description on the types and techniques will be dealt in detail when enhancements in model based testing are discussed. Thus the term model based testing in system testing perspective focus on the usage of state machine diagrams for test design phase. As such the current industrial usage of these state diagrams cannot be used for direct execution, instead one need to have low level test cases.

EXISTING GAPS IN MODEL BASED TESTING

Current model based testing is defined to derive low level test cases from the defined models. So it is a two step process of defining a model first and then to derive test cases for further execution. This two stepped process may be necessary for an application that actually demands such a low level visibility for quality execution. But for all other conditions or functionalities that really doesn’t makes more meaningful to have double edged documentation increases the documentation time only. For short run programs, one cannot afford huge investment of incurring tools for generating test cases with the use of models, as that will give only a less return on investment (ROI). Mostly from the work observed by applying the existing MBT for web applications this is found to be true. But the fact is also accepted for few areas where there is a need for drilling to low level test inputs. Also existing model based testing doesn’t addresses on the fact of linkage of having a single test case document, if models are used for test case generation. Since models do not only contribute to overall test cases, the establishment of relationship with other relevant test cases with the generated models is found necessary so that test design is not duplicated and also coverage is not missed out. Most importantly, a specific technique for a specific domain so as to derive models as universally accepted standard is yet in its infancy.

A formal test model generation algorithm is insisted that will comprise of both black box and white box models. This also lags in clearly specifying on when to go for black or white box models. As most of the models are to be provided as an input to some automated tools for test case generation that are normally recommended for existing model based tools, the difference in models generation techniques varies with the usage of tools. At present, many commercially available tools in regard with their way of implementation for deriving the models, expect the tester to be 1/3 developer, 1/3 system engineer, and 1/3 tester. To seek these skilled testers is difficult in current industry or the budget to hire such testers is costly. This can inturn be remarked that the developed technology for model based tools does not address the competency of majority of its end users. Because of the above described gaps in existing model based testing, there is need to enhance for techniques that will be adopted readily by testers with the goal of improved quality and productivity.

ENHANCEMENTS FOR EXISTING MBT

The enhancements to be derived are inline with the order of listed drawbacks as described in the previous section. Even before outlining the enhancements, the current model based testing is described using state diagrams so that exact mapping to overcome on the drawbacks can be better understood. Fig 2 represents current industry practice on the usage of model based testing with finite state transitions. Here every transition contains input event and a next state. A transition can also define outputs and actions. Thus finite states operate by traversing from each of the states defined for the system.

![Finite State Machine for Single Phone System](Source:https://goldpractice.thedacs.com/practices/mbt/)

**Fig 2: Current usage of MBT**

From the described states, a form of truth table or low level test cases are generated to perform test execution and update test results. With these basics, enhancements that are envisioned for current model based testing are detailed with current usage and business gap.

*Enhancement 1 – Objective of State flows*

The current models are not directly used for test execution because of the fact that depicted figure doesn’t depict the objective of the flow (as the flows are interrelated without a single point of termination), its input and a terminated output as expected output for test execution. The first enhancement would be providing objective of the state flows with its relevant test input to test.

*Enhancement 2 – Requirements Traceability*

The current state flows do not have any backward traceability with requirements to justify on test coverage, though the derived states are claimed to be derived from requirements and to some extent with application design too. Thus enhanced model will aim to provide requirement traceability in the state diagrams.
**Enhancement 3 – Linkage with relevant states**

The current model just depicts the extended flows from the output state of another flow. But in reality, the tester need to know on what input condition another flow is triggered and how to make a priority to run the test cases. The enhanced model will depict on separating the flows but at the same time to maintain linkage with related flows so as to provide inputs for the order of execution and also to find out dependent states.

**Enhancement 4 – Test state ID & Priority**

The current state flow diagrams does not support for maintaining test state identifiers (ID’s) & priority that will support for test results and subsequent metrics derivatives. Thus enhancement will be made to capture separate ID’s for test states and also to priority the state flows.

**Enhancement 5 – Expected Output**

The current state flow diagram represent only the subsequent transition in terms of functionality and does not really support for what are the other expected output conditions for that transition in testing perspective. In the given state flow example the state transition for ‘Dial Tone’ happens from ‘On Hook’ or ‘Talking’ with different trigger inputs. Though the tester may know that ‘Dial Tone’ is the expected transition for those input states and conditions, he may not be aware of other conditions like ‘type of the dial tone’ & ‘duration of dial tone’. This gap when analysed for critical web application functionalities will make it more meaningful and hope this specific point could be better understood by an experienced tester. So, the envisioned model based testing will modify the definition of output state transition incorporating the gap in terms of expected output.

**Enhancement 6 – Linkage with conventional test cases**

Though this is not an actual enhancement in current model based testing, this specific point of linkage with other test cases need to be taken care when implementing enhanced model based testing. As envisioned state diagrams are executable in nature, these state flows should have formal linkage with other conventional cases in terms of data definitions that describes field length, type, behaviour etc., integration checklist and generic cases. This will be detailed when describing the complete vision of enhanced model based testing with executable state diagram approach.

**MODEL BASED TESTING WITH EXECUTABLE APPROACH**

Collating all major six enhancements, executable model based testing is described using simple ‘Login’ example. The terminologies used in this approach are initial state, input conditions, trigger, state transition, extended state, test state ID, test state priority, state-integration map Id, state-data definition map id, state-generic cases map id. The brief description of this terminologies are listed below and this when traced with further example will provide still more insight.

<table>
<thead>
<tr>
<th>Terminologies</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial State</td>
<td>Describes the condition of the state without being disturbed by input conditions or trigger</td>
</tr>
<tr>
<td>Input Condition</td>
<td>Describes the inputs to the system that is still in its initial state</td>
</tr>
<tr>
<td>Trigger</td>
<td>Describes the act that actually transition the state for the provided input condition</td>
</tr>
<tr>
<td>State Transition</td>
<td>Describes the altered state on subjecting the initial state with relevant test inputs and trigger</td>
</tr>
<tr>
<td>Extended State Mapping</td>
<td>Relates with other internal or external initial states or triggers from the transitioned state</td>
</tr>
<tr>
<td>Test State ID</td>
<td>Unique identifier to each of the captured state flows</td>
</tr>
<tr>
<td>Test State Priority</td>
<td>Identification of priority of each of the captured state flows inline with critical to quality</td>
</tr>
<tr>
<td>State-Integration, State-data definition, State-Generic cases Map ID</td>
<td>Mapping identifiers for the captured state flows with integration cases, data definition and generic cases</td>
</tr>
</tbody>
</table>

Table 1: Terminologies in MBT- Executable State Diagram Approach

**DEMONSTRATION OF EXECUTABLE MODELS**

For a simple functionality – ‘Login’, that authenticates and authorizes the user based on the entered credentials, first level understanding on initial and transitioned state is described. The initial state of the web application being a login page that has provision for entry details, the possible output states for the described initial state (irrespective of taken care in implementation) can be a page loading successfully as per user privileges, page loaded successfully with incorrect user privilege, login page retains with specific error and appearing of an unexpected error page. This is depicted below for simplicity in Fig 3.
Further to drill on the exact enhancement in model based testing with all the terminologies described in previous section, the login functionality is enhanced with a feature of having a condition check if they are first time login user. If so it will re-direct it to a form filling page, else it will re-direct it to application home page as per user privileges. The enhanced executable model based testing with executable state diagrams are depicted in Fig 4.

**Fig 3: Rough Extraction for MBT**

![Login page with entry details](image)

**Fig 4: MBT – Executable State Diagram**
From the above enhancements, it is seen that the initial state and input conditions are based on the conventional way of writing the test cases. But the readability and maintainability that it offers makes the difference. As described, the way of converting the ‘output states’ as captured in Fig 3 is different from ‘state transition’ that is captured in Fig 4.

For better illustration the second output state of ‘Page loaded successfully with incorrect user privilege’ is considered. For executable state transitions, this thought process on output state should be refined in the form of expected output that is inline with functional specification. In this context, the rough thought process of illustrated output state gets mapped into state transition as ‘Application should not allow the user to login by displaying specific error message display’. As explained, ‘State transition’ parameter of executable model based testing should emphasize on application behaviour inline with functional specifications, database/or any external dependent component behavior for that input trigger and behaviour in terms of usability perspective.

Apart from this enhancement, the enhanced model also has extended states which depict the entire usage flow that cannot be achieved using conventional test cases. A simple scenario that is captured as a part of extended state transition is ‘user logging with valid credentials and logging out with a further flow of re-login for unauthorized privileges’. As stated every state transition and extended state transition have test state Id’s that captures unique input and output conditions. For many cases, one or more triggers of same input can lead to same state transition. In this context one increases the coverage with less documentation. The captured states Id’s are mapped back with requirements to satisfy the traceability matrix criteria. Against each and every test states, the results are updated for ‘Pass’, ‘Fail’ criteria and thus satisfying the conventional process so as to avoid low level test cases. Thus the conventional way of writing 48 test cases for this functionality is reduced to 14 test states including state transition and extended transitions. The details on further enhancements of test states and results update is given below.

<table>
<thead>
<tr>
<th>Login: Req Id: CBAM Req_Login 01</th>
<th>Test Execution Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>CBAM_ST_Login 1</td>
<td>Result</td>
</tr>
<tr>
<td>CBAM_ST_Login 2</td>
<td>Same</td>
</tr>
<tr>
<td>CBAM_ST_Login 3</td>
<td>Same</td>
</tr>
<tr>
<td>CBAM_ST_Login 4</td>
<td>Same</td>
</tr>
<tr>
<td>CBAM_ST_Login 5</td>
<td>Same</td>
</tr>
</tbody>
</table>

Fig 5: MBT – Enhancements

Final set of enhancement includes a set of precautionary steps to arrest the gap because of the enhanced model based testing as depicted in Fig 6.
The main missing gaps that are addressed as a part of enhancements are the test data for valid and invalid details for data definitions, integration cases and generic cases specific to each of the captured states. So, for every state flows it is assumed that there will be relevant data definitions, integration and generic cases and these are mapped to a different sheet with unique mapping ID. Thus as a whole the enhanced executable model based testing provides better opportunity for optimizing test design and also inherits the advantages of basic model based testing. The detail on the list of advantages will be dealt in different section.

**MODEL BASED TESTING – SUPPORTING TYPES AND TECHNIQUES**

As with conventional test cases that are supported by different types and techniques for better coverage, model based testing is also aided by some types and techniques. But the base theme of deriving the models remains constant. The enhanced executable model that is described in previous section falls into "finite state machine" diagrams. The other type under model based testing is state transition table which is a form of truth table that is applicable for practical usage when there exists logical reasoning for more than one output variables as below.

<table>
<thead>
<tr>
<th>Application Status</th>
<th>Action</th>
<th>Status display for submitter</th>
<th>Status display for Admin</th>
<th>Any mail generation</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>R lust B C 01</td>
<td>R lust B C 699</td>
<td>Parent ID is not entered</td>
<td>Yes</td>
<td>No names</td>
<td>Vs</td>
</tr>
<tr>
<td>R lust B C 01</td>
<td>R lust B C 670</td>
<td>Parent ID is not entered &amp; saved</td>
<td>In progress</td>
<td>No names</td>
<td>Vs</td>
</tr>
<tr>
<td>R lust B C 01</td>
<td>R lust B C 871</td>
<td>Parent ID is not entered &amp; submitted</td>
<td>In progress</td>
<td>In progress</td>
<td>Vs</td>
</tr>
<tr>
<td>R lust B C 01</td>
<td>R lust B C 872</td>
<td>Parent ID is not entered &amp; submitted pending for a long time to approve</td>
<td>In progress</td>
<td>In progress</td>
<td>Vs</td>
</tr>
<tr>
<td>R lust B C 01</td>
<td>R lust B C 873</td>
<td>Parent ID is not approved</td>
<td>Approved</td>
<td>Approved</td>
<td>Vs</td>
</tr>
<tr>
<td>R lust B C 01</td>
<td>R lust B C 874</td>
<td>Parent ID is not approved &amp; action taken less than 1 day for submitter</td>
<td>In progress</td>
<td>In progress</td>
<td>Vs</td>
</tr>
<tr>
<td>R lust B C 01</td>
<td>R lust B C 875</td>
<td>Parent ID is not approved &amp; action taken</td>
<td>In progress</td>
<td>In progress</td>
<td>Vs</td>
</tr>
<tr>
<td>R lust B C 01</td>
<td>R lust B C 876</td>
<td>Parent ID is not approved &amp; action taken</td>
<td>In progress</td>
<td>In progress</td>
<td>Vs</td>
</tr>
<tr>
<td>R lust B C 01</td>
<td>R lust B C 877</td>
<td>Parent ID is not approved &amp; action taken</td>
<td>In progress</td>
<td>In progress</td>
<td>Vs</td>
</tr>
</tbody>
</table>

Be it finite state machines or state transition diagrams, the supporting techniques falls into one or more of the given described categories.

**Theorem proving**

This technique approaches the system as modeled by a set of logical expressions specifying the system's behavior. The model is partitioned into equivalence classes over the valid interpretation of the set of the logical expressions describing the system under test.
Symbolic execution

Symbolically defining data that focus program path to be executed (instead of actual data values, variable names are used). Thus all variable become string variables

Event-flow model

Deriving test states that represents events and event interactions like UI & any external components

Model checking

This defines a model of the system under test and a property to test to the model checker. E.g.: one referring "Use case diagram" for deriving state flows.

EXECUTABLE MODELS – DEPLOYMENT CONSIDERATIONS

This section will provide insight on when to use executable models pertaining to their value addition for test design that in turn assists execution phase. With the demonstration of executable models it is understood that it requires system engineering capability to derive the models. The readability increases if one use any visual drawing tools. In this case, the start up effort for executable models for beginner will consume time till they are comfortable in system engineering & use of manual aiding tool practices. Excluding this effort, there is a need of delta additional effort when compared with traditional test design approach. So it is always necessary to think over on few aspects of return on investment in terms of coverage & productivity front before going with executable models.

- Check if there are any workflow functionalities and if so its magnitude comparing with overall application functionality
- Check if there are any re-usable frameworks in terms of workflow between the modules.
- Check if there is any logical interaction between applications/external components.
- Check if functionality of application deals more in terms of logical context.
- For enhancement/maintenance phase, consider the frequency of changes within the modifications.

EXECUTABLE MODELS – BENEFIT’S DEMONSTRATED

The executable model based testing is applied for 3 full lifecycle projects related to trademark maintenance and banking application along with an aero project that calculates the optimum distance for gear with set of input conditions in place. It is also applied for 3 enhancement projects related to virtual network and finance based applications. In these applied domains (as such is not restricted) the executable model based testing has revealed more than that is expected as initially foreseen value additions or advantages.

1. The foremost advantage it offers is a better readability and coverage of high critical business scenarios. For few of the comparative modules/ functionalities that were translated from conventional test cases, it was observed that, around 30-40% additional/linking business scenarios are able to cover using this approach. In terms of readability the metric used is ‘review effort’ and it was seen that the decrease in effort is around 40-50% than reviewing conventional test cases.

2. Also it was seen that linkage factors with integration cases has increased the knowledge on application understanding and use of state transition table has helped to identify requirements gap at the initial stage itself. This increase in high severity requirement review comments are found to be 60-70% more than practicing a conventional requirements walkthrough and test design. It can also be noted that, executable model based testing doesn’t need a separate review effort as it is covered as a part of system engineering itself.

3. One of the hidden treasure in executable model based testing is that it provide assistance to application design that inturn helps in “test driven development”. Few of the missing details that are found using executable models is related to interface diagram, sequence diagram and data modelling details.

4. The most important feature is that it reduces maintenance effort because of its executable and readable nature. It also helps to identify the regression flows in negligible time. It is found that the maintenance effort for any updates in test design has reduced to an extent of 60-70%. The main increase in reduction is it doesn’t need a separate effort for detail requirements understanding of earlier version because of the nature of executable model based testing.

With these qualitative benefits described for the set of applications where the executable model based testing is deployed, the details on metrics for these projects are given below.
**Table 2: MBT Metrics**

<table>
<thead>
<tr>
<th></th>
<th>Project 1</th>
<th>Project 2</th>
<th>Project 3</th>
<th>Project 4</th>
<th>Project 5</th>
<th>Project 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maintenance Eff (hrs)</td>
<td>12</td>
<td>4</td>
<td>6</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Review Eff (hrs)</td>
<td>17</td>
<td>16</td>
<td>3</td>
<td>7</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>R/Reuse Cases</td>
<td>50</td>
<td>20</td>
<td>40</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>R/Test cases</td>
<td>500</td>
<td>60</td>
<td>120</td>
<td>30</td>
<td>50</td>
<td>12</td>
</tr>
<tr>
<td>Design comments</td>
<td>60</td>
<td>20</td>
<td>12</td>
<td>0</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>Review comments</td>
<td>470</td>
<td>120</td>
<td>50</td>
<td>60</td>
<td>40</td>
<td>24</td>
</tr>
<tr>
<td>R/Requirements</td>
<td>1200</td>
<td>170</td>
<td>330</td>
<td>80</td>
<td>150</td>
<td>40</td>
</tr>
</tbody>
</table>

Project 1 till 3 are full life cycle projects and other projects are of enhancement types. Also Project 1 and 2 followed agile development model. From the above data it is clear that the requirement review comments are huge while adopting the model based testing approach. Also there are few design comments captured (though explicitly doesn’t target for) that also contributes to the overall reduction in project life cycle effort. The test states covered doesn’t include any integration or data validation count. Also the maintenance effort calculated here is based on the effort required to identify the relevant test cases for change, making changes and for identifying regression cases for further testing.

**CONCLUSION**

The use of enhanced model based testing is found to be appropriate for workflow related application functionalities and for areas where there are more logical requirements is involved. The enhanced model also compliment all the necessary conventional features along with its eminent benefits as described for basic model based testing. As for any model based automated tools, it is required to have models as inputs, the described idea on model based testing will provide information store for the same. Since it is an agreed upon fact that the tools cannot be used for the generation of models and the executable generation of models can be used where there is no significant dependency in low level test cases. Also the paper has provided an in depth knowledge with practical examples of deriving the models that can be used directly for test execution.

**REFERENCES**


[3] Position Paper: Model- Based Testing, Mark Utting-The University of Waikato, New Zealand

**Biography**

Ms. Anna has expertise in functional, performance, compatibility, automation and white-box testing mainly into the domains of Aerospace, Finance and Banking systems. She is a CSTE certified (conducted by QAI USA) and specializes in test estimation methodologies, tools analysis and in test process improvement areas. She is also Six-sigma and green belt certified professional. She has contributed white papers for International testing conferences like QAI, STeP AUTO and IEEE sponsored conference in topics related to test estimation, test process improvement areas, webservises and cloud computing technologies.