Anatomy of Model Based Testing

Keerthi Kumar Narayan, Sharan Kumar Paratala Rajagopal

Abstract*: *In a typical Software Development Life Cycle (SDLC), the software testing life cycle consists of reviewing of the requirements, test planning for design, development and execution. Test designing phase is considered as the most vital and foundational in deriving the test cases against the software or the application to be validate. The known fact is that in order to derive an effective test suite generally consumes a lot of manual efforts and good amount of expertise as well. [1] When the testers validate an application for its correct and required behavior, then that system is known as System Under Test (aka SUT), the most common term used in software testing process. Since, this is purely based on a manual approach and testers may not be able to validate all the possible and required scenarios, there may be risk of putting the system for validation. Because, the application may break under a particular use case. This can be overcome by applying Model Based Testing (MBT).

Keywords: MBT, Model Based Testing, Testing, Model Based Design

I. INTRODUCTION TO MBT

MBT is a technique where the test cases are created by a model which depicts the functional aspects of the SUT. This will use models to create test cases for both offline & online testing. These test cases play a vital role in detecting significant errors compared to that of the manual test process.

Basically, it has been accounted that testing process involved more than 50%-60% of the cost in a project in order to confirm the readiness of the system to be pushed into the production environment. In order to achieve this, all the intended behavior from the specifications document needs to be brought into a real time application and this will be finally ensured with the fine-grained level of testing the application. The requirements specified in specifications requirements documents will be always tends to have issues, incomplete, ambiguous and confusing as well. For a tester it will always be a challenging and difficult to derive test cases with such documents. Effective usage of an automation test framework for automating the creation of the test scripts using a model-based approach will significantly help in reducing the cost and the manual efforts to be made [5]. This white paper basically aims at detailing about an automated testing framework known as "Model Based Testing". Making efficient use of an automated testing framework will be having an ensured increase in test speed, accuracy and will result in the reduction of the maintenance cost and lowers the risks. Model Based Testing (MBT) is a modern approach in the automation test phase. MBT is an approach where test cases to be executed are automatically derived or generated from the models. These models are nothing but the system under test (SUT) that represents the strategy of the testing phase. In simple, a model is an abstraction of a real-world function. MBT approach is considered as an inception towards the creation of modern automation test frameworks and an acceptance criterion for the test design, development, execution and maintenance activities. In test development, MBT is not just that can be only used to design and develop the test cases for the manual execution. MBT can be easily used to automate these test cases to be executed completely in automated manner. Models are created from SUT requirements and behavior reported about the application in the functional specs. Model typically helps to gain better understanding about the behavior of the application compared to the actual software specifications. MBT aims at creating a no. of test cases in a very short span of time accommodating almost all the possible scenarios which cannot be verified when testing is performed manually. MBT also promote traceability by mapping the test cases properly with the requirements provided in the specifications document.

It has been observed that MBT establishes a systematic way of testing. Wrong understanding of the requirements will always lead to incorrect modelling of the system and as a result application will be faulty. A model can also be considered as the blueprint and halfway representation of SUT. Once the model is created from the specifications and requirements, test sequences are derived from the same and they are known by unique test suite. The role of test sequence is to manage the control the SUT, where all the required conditions will be verified to match the model that is derived. A typical representation of a MBT flow is shown below:

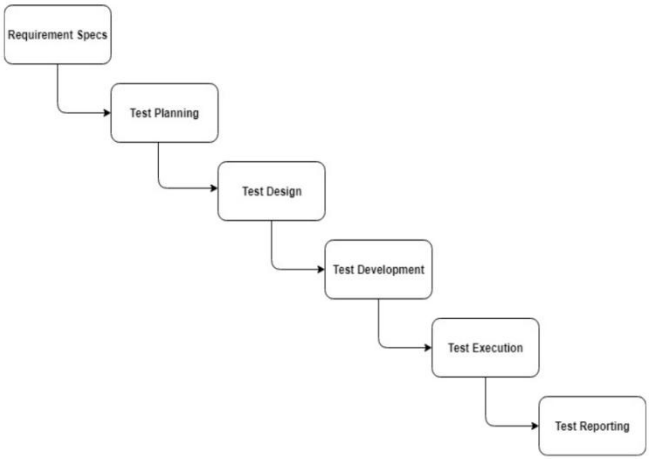

Figure 1 - Representation of MBT Flow

Revised Manuscript Received on June 22, 2020.

Keerthi Kumar Narayan, Senior Member Technical Staff, Bangalore, India. Email: n.keerthikumar85@gmail.com

Sharan Kumar Paratala Rajagopal, Senior Manager, Capgemini America, Inc., Dallas, Texas, USA. Email: prsharankumar@gmail.com

What is a Model?

The fundamental idea is that from a formal or semi-formal model (e.g. transition system, UML State Machines, class diagrams extended with constraints, etc.) complete test cases (input and expected output pairs) can be generated. There is an extensive research in this field, but the technology has matured enough that nowadays there are commercial tools and industrial applications [3]. A model is defined as a description of a system's behavior. Behavior can be described in terms of requirements, inputs, actions/events, conditions, output and flow of data. The model should be derived in such a way that it should be practically reusable, shareable and should possess the detailed information of the SUT. Models are created from SUT requirements and behavior reported about the application in the functional specs. Model typically helps to gain better understanding about the behavior of the application compared to the actual software specifications. MBT aims at creating a no. of test cases in a very short span of time accommodating almost all the possible scenarios which cannot be verified when testing is performed manually. MBT also promote traceability by mapping the test cases properly with the requirements provided in the specifications document. It has been observed that MBT establishes a systematic way of testing. Wrong understanding of the requirements will always lead to incorrect modelling of the system and as a result application will be faulty. A model can also be considered as the blueprint and halfway representation of the SUT. Once the model is created from the specifications and requirements, test sequences are derived from the same and they are known by unique test suite. The role of test sequence is to manage the control the SUT, where all the required conditions will be verified to match the model that is derived. A typical representation of MBT flow is shown below:

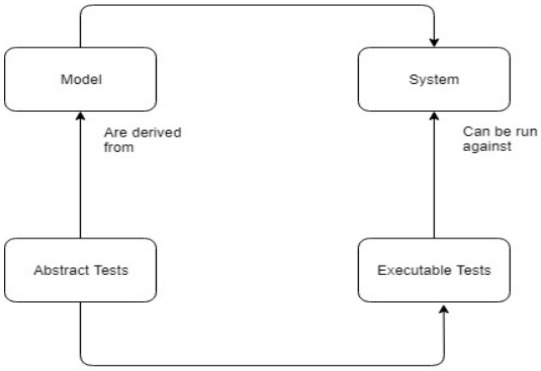

Figure 2 - Representation of MBT Flow

In the traditional practice, when planning the test cases for the software testing phase, testers read through the software functional specifications to understand what is to be tested. Then the test cases will be derived for each specification identified for the software/system. Depending on the complexity of the SUT and the model the number of paths can grow larger, due to the fact of huge number of configurations made in the system.

II. BACKGROUND

Test automation is phase in the software testing where it uses a software separate from the software being validated to manage the execution of the tests and comparing the actual results with the predicted ones. Though a test automation can neatly automate some of the repetitive tests, but all the required tasks needs to be put in place prior to automating the tests. Also, the automation testing plays a vital role in the continuous delivery and continuous integration. According to Gartner, organizations spend nearly 20% of their SDLC time in system testing and defect removal, and testing is typically 15%-20% of IT budgets [2]. It is vital that every feature of the software must be tested thoroughly in order to ensure the system is working as expected with regards to the requirements defined. Software testing is a process which is very iterative and consumes significant portion of the software in terms of budget, time and efforts. The critical thing is that testing the application during the subsequent releases that happens most of the times. Also, there could be of a very high chances that issues may be reported, and this too requires validation after the fix is delivered. These are the classical scenarios where manual testing process will be highly expensive due to so many adhoc testing needs to be performed.

TABLE - 1: ABBREVIATION

Acronym	Full Form
SUT	System Under Test
SRS	System Requirements Specifications
MBT	Model Based Testing
GUI	Graphical User Interface
BRD	Business Requirements Documentation
UML	Unified Modelling Language

III. CHALLENGES WITH MANUAL TESTING

With the current trend in the field of testing, the test design and test automation are a specialized skillset and most of the times it is observed that these specific and specialized skills result in shortage with the supply. Below are some of the key challenges identified with respect to the manual testing [4].

A. Test Design

Designing the tests involves a good expertise and detailed knowledge on the domain of which the application which is to be tested. Most of the times only few testers will be available who ensures to follow the standards in the test creation and execution by iteration going through the test design with a detailed attention.

B. Test Automation

Since automation is a process where basic coding skills are imperative and vital. Finding the availability of testers with developer mindset and enough knowledge on the test script automation is truly a challenging one.

C. Poor test case design and limited coverage

It is admitted that the manual test case design is a time-consuming process. If the requirements are poor and incomplete, the same gets translated into a poor test case design where there will be a high chance that may lead to design flaws. Poor testing also leads to over-testing of the same features of the application. With regards to the coverage, testing usually marks only limited amount of functional test coverage as most of the times testing will be done on adhoc basis.

D. Testing the complete application

It is a well-known fact that it is almost impossible to validate a complete application thoroughly and certifying that application will be a bug free for a lifetime. It is since every permutation and combination of the manual test cases can be tested fully.

E. Incomplete clarity on the process

There are some myths being told that testers should only follow the standards and principles defined by the company despite being those processes may not be really required to be followed which will surely result in the incomplete testing of the application.

F. Waiting for the test data

The necessary test data is difficult to be made available whenever the testers in need of it. This data constraint enforces the testers to wait for data to become available. Even refreshing of the data sometimes consumes more time and ends up the testers to wait for longer times which easily consumes more testing efforts.

G. Relationship between developers and testers

This is one of the most challenging part for any tester as it is expected that the tester should be intellectual enough to handle an irrefutable relationship with the developers and ensure to complete all the testing related tasks along with which it includes a good communication and troubleshooting skills.

IV. HOW MBT WORKS

In general, the model is translated to or read as a finite state automaton or a state transition system. These automata represent the configurations done to the SUT. In order to find and generate the test cases, the automata is searched for executable paths. Any possible execution path can be made as a test case.

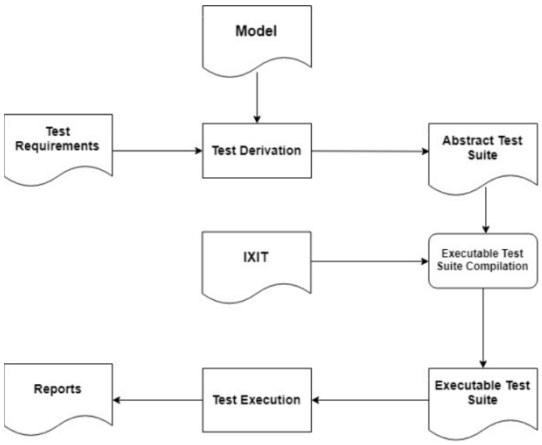

Figure 3 – MBT Flow Execution

V. TYPES OF MBT

Offline - In this approach the test suites are generated before the execution of the same.

Online - In this approach, the test suites are generated dynamically during the test execution

VI. MBT TOOL APPROACHES

A. Keyword Driven

This testing approach separates the test design and its execution process.

This works technique by specifying keywords like click button, open, close and enter, etc.

These keywords are then transformed to executable test scripts.

Table - 2: Login Flow with Keyword Driven

Model	Control	Event/Action	Arguments
Login	Browser	Open	www.mywebapp.com
Login		Enter	Username, Password
Login	Login Button	Click	Login
Login		Verify	Homepage
Login	Browser	Open	www.mywebapp.com
Login		Enter	Invalid credentials
Login	Login Button	Click	Login
Login		Verify	Error message shown

B. Data Driven

This testing approach is used when a single test needs to be validated across multiple set of data.

Test data will be read from external data sources like excel, csv or xml.

MBT integrates test data with different actions, transitions and states.

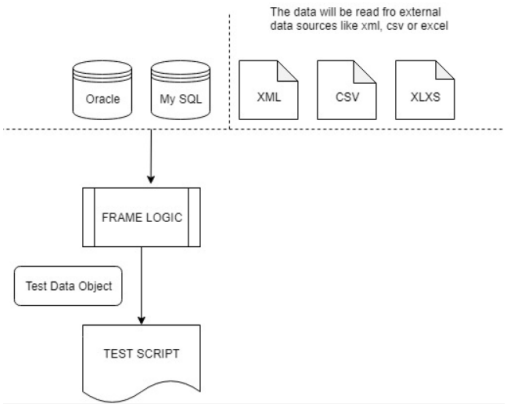

Figure 4 – Data Driven Approach

TABLE - 3: LOGIN FLOW WITH DATA DRIVEN

Username	Password	Result
ValidUser	ValidPassword	Pass
ValidUser	InvalidPassword	Fail
ValidAdminUser	ValidPassword	Pass
ValidAdminUser	InvalidPassword	Fail

C. Structured Drive

This testing approach allows structured and modular view of the entire SUT.

It represents the systems in the form of functions with the use of control structures like if-else, for, switch etc.

These functions are then used as hierarchical fashion to build larger tests.

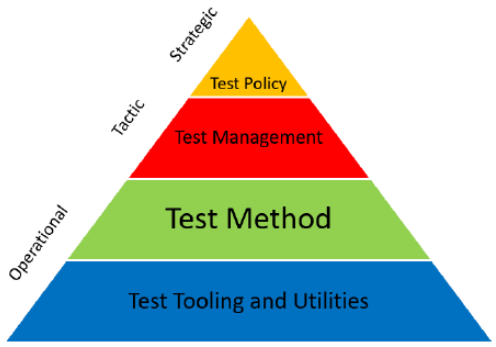

Figure 5 – Structured Driven Approach

D. Hybrid Driven

This testing approach is the most commonly used and combination of the above-mentioned approaches.

Most of the projects will be having an automation projects with this approach.

Below is the simple example of combining keyword and data-driven approach for a simple login page.

TABLE - 4: LOGIN FLOW WITH HYBRID DRIVEN

Module	Control	Action	Arguments
LoginPage	Browser	Open	http://www.mywebapp.com
LoginPage		Enter	Username, Password
LoginPage	Browser	Click	Login
LoginPage		Validate	Homepage (Successful Login)

VII. BENEFITS

The model-based testing basically kick starts with the specifications to ensure that the QA process is indulged from the early stage of the project itself rather than waiting till the phase where development phase is started. There will be a direct investment in the design and developing a more reliable, robust and easily maintainable testing suite (with an automated testing framework) for a given application which will act as a system under test (SUT). The primary advantage of the MBT with the creation and usage of models helps to the best to identify the bugs even much before the code for the same is developed. Below are some of the key benefits identified with regards to the effective usage of MBT for developing an automated test framework.

- Easy maintenance of the test suite
- Cost reduction
- Early detection of the bugs
- Test coverage improvement
- Time saving
- Faster test automation
- Good coverage
- Reduced cycle time

VIII. CONCLUSION

Model based testing is one of the finest approaches in the software testing field. Any automation test framework build using model-based testing basically eventuate with the design of the test cases and execution details and then detailing about the manual test procedures. These manual test procedures then make effective use of the automation test framework in order to deliver modularity in the test case design. Once, all these tasks are completed, these tests will be then integrated with the automation test tools to execute them automatically [6]. The generation of the test cases in an automated manner has proven significantly in increased effectiveness in finding the bugs relevantly in the SUT. This directly ensures in the increased quality of the product even in the circumstances of any unplanned project releases. Also, the manual effort required in designing, developing and maintaining of the test cases has seen a significant reduction in the recent times.

MBT greatly scales when there is a shortage of resources for the testing phase and really helps to deliver the tests at the earliest so that an automated test framework will be in place starting from the very first test cycle.

REFERENCES

1. Eckard Bringmann, Andreas Krämer PikeTec GmbH, Germany Eckard.Bringmann@PikeTec.com, Andreas.Kraemer@PikeTec.com https://files.piketec.com/downloads/papers/Kraemer2008-Model_based_testing_of_automotive_systems.pdf
2. Royal Cyber, 20-Feb-2019 https://www.royalcyber.com/wp-content/uploads/2019/03/Flexible-Approach-to-Test-Automation-with-TEAF-WhitePaper.pdf
3. Zoltan Micskei – Model Based Testing [MBT] http://mit.bme.hu/~micskeiz/pages/modelbased_testing.html
4. Greg Sypolt, 2018 https://saucelabs.com/blog/the-challenges-and-benefits-of-model-based-testing,
5. Shafique, Muhammad & Labiche, Yvan. (2010). A systematic review of model based testing tool support. Shafique, Yvan Labiche, May 2010

AUTHORS PROFILE

Keerthi Kumar, a passionate java/j2ee developer who is eager to study, upskill & adapt himself to latest technologies related to server-side programming. Keerthi Kumar has worked in various domains including insurance, life science, e-governance, telecom, retail, travel, sales and has gained extensive knowledge of both functional and technical aspects. Keerthi Kumar also contributes to some of the major learning web sites by publishing tech articles and tutorials. He is a regular participant in hackathon and coding events. Keerthi Kumar currently serves as a Senior Member Technical Staff at Amadeus Software Labs India Private Limited, Bangalore.

Sharan Kumar Paratala Rajagopal is a Senior Manager with Capgemini America, Inc. having 14+ years of design, development and architecture experience. Specialized in Java/J2EE, Integration methodologies, Guidewire Product, Data Analytics, AI and Cloud technologies. Vast domain experience in Public Services, Hospitality and Property & Casualty Insurance. Also contributed multiple technical articles to major Dev communities.