ENGINEERING AND R&D SERVICES

HCL

WHITE PAPER on Test Automation Framework Using MBT

Dec 2013

e2 Engineering experiences

Acceleration | Value | Technology

TABLE OF CONTENTS

Test design and the subsequent test development tasks can both be automated using model-based testing.

While test automation is used to automate test execution, model-based testing can be used to automate even the process of test automation.

Abstract

The software testing lifecycle consists primarily of requirement review, test planning, test design, test development, test execution and test reporting. This white paper looks at applying model-based testing for automating test design and test development phases. Within the test development phase, this looks at applying MBT more specifically to the activity of developing test scripts for automated execution. This sets out the steps involved in building a test automation framework using MBT.

Test design is an indispensable and foundational phase for software testing. This involves analyzing either the requirements or the specifications and coming up with test cases against which the software will be validated, to determine if the requirements are met or the specifications are complied with. This requires human effort.

Test development involves developing the test procedures (manual or automated) that will be repeated on the software being tested. Typically these tests are executed manually and involve human effort. Automating software testing is an option, and involves developing test scripts using scripting languages such as Python, JavaScript or Tcl, so that tests can be executed by computers with minimal human intervention and attention. Whether writing the manual procedures or the automated procedures, both involve effort. The additional effort expended in coding automated test procedures is the

initial investment one makes for the recurring savings anticipated in successive regression cycles.

Test design and development together can be automated to reduce human effort and save cost. Developing test scripts (manual and even automated) can be automated using model-based testing. A model is first created to capture the behavior of the system under test. An MBT tool then parses this to create manual test scripts. The tool can additionally generate automated test scripts for automating test execution and can integrate with popular test automation frameworks and tools.

Automating the creation of both manual test scripts and automated test scripts using a model not only saves effort and thereby cost, but increases coverage and also significantly reduces the time-to-market. It is imperative that software vendors do not compromise on software quality, and therefore testing cannot be avoided. Automation provides the lever to cut cost and time without compromising on quality.

Abbreviation

Sl. No.	Acronyms	Full Form
1	BRD	Business Requirements Document
2	GUI	Graphical User Interface
3	MBT	Model-Based Testing
4	SFS	Software Functional Specification
5	SRS	Software Requirements Specification
6	SUT	System Under Test
7	UML	Unified Modeling Language
8	QML	Qtronic Modeling Language
9	QTP	Quick Test Professional
10	QA	Quality Assurance

MBT usage spans a wide range of application stacks, software processes, application domains, and development organizations.

These include open source and commercial providers; embedded, transaction processing, and communications applications; C/C++, Java, C#, and other programming languages; and process models including none, Agile, incremental, and V-model. [1]

Background

Several advancements have happened in the field of software testing. This white paper will look at two specific advancements and bring them together to solve a common problem software vendors face when it comes to testing their products.

Requirement capture through models waas introduced more than two decades ago. Howeve, models for the purpose of software testing have been talked about only over the last decade. IBM, MathWorks, ConformiQ, TestOptimal and several others provide tools that let users create models and process them further for creating test cases and test scripts.

Test automation practices, too, have evolved over the last decade. While linear test scripts gave way to structured ones that use test libraries, data was still embedded and modifying test cases required programming knowledge. Newer approaches like data-driven and keyword-driven testing source the data and even the directives from external files, insulating the test designer from the complexities of scripting.[2]

These advancements can be powerfully combined to reduce the effort and time required for software testing, allowing software vendors to be very competitive and aggressive.

Market

Testing occupies a significant portion of the software development lifecycle in terms of both cost and time. Every feature has to be tested, not only when it is introduced for the first time through a release, but every time a subsequent release carries it with or without enhancements to it. Consequently, regression test cycles are long, and tests within the cycles repeat every month or quarter depending on the release frequency, and often require modification. Manual testing in such a scenario becomes prohibitively expensive, and software vendors are increasingly looking at test automation to reduce the recurring cost (of labor) and time. [3]

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.

Per a NASSCOM Report, in FY 2010 the global testing market was worth USD30 Billion, and offshore testing revenues amounted to USD10.5 Billion, of which India accounted for USD3.2 Billion. By FY 2020, the global testing ,arket is expected to reach USD50 Billion, and India is expected to account for USD24-28 Billion, capturing a lion's share. [4]

Test design and test automation are specialized skills that are always in short supply. Demand for these skills often exceeds supply.

Challenges

Test design and test automation are specialized skills. These skills are always in short supply.

Test Design

Test design involves test case creation, and requires sound knowledge of the domain in particular, and of software development and execution in general. Very few testers who can follow a test plan and execute test scripts like going through the rigors of test design that require great discipline and attention to detail.

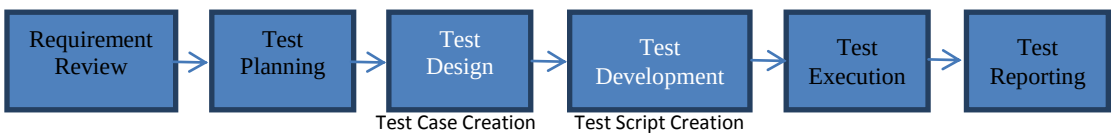
Test Automation

As test automation is about automating test execution by developing automated test scripts, it requires coding skills. Someone proficient in coding would rather be a developer than a tester, as the role of a developer has been a highly fancied role compared to that of a test automation engineer. The availability of resources with a developer mindset trained in one of the languages used for test automation scripts is a challenge.

Solution

Model-based testing is beginning to find acceptance as a methodology for test design and test development (refer to Figure 1).

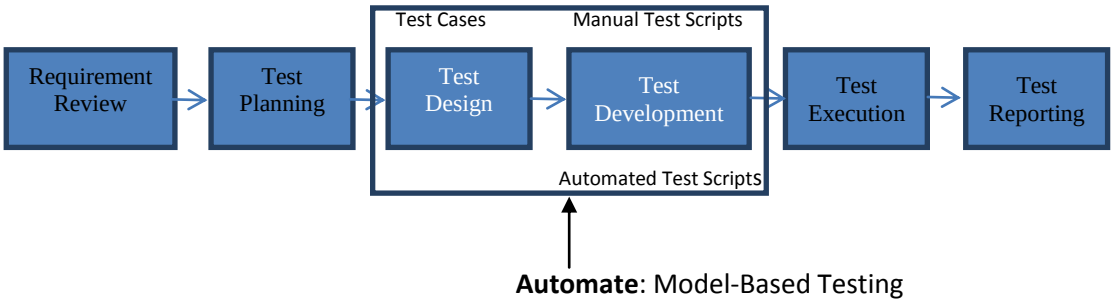
Figure 1: Test Life Cycle



(Manual & Automated)

In test development, MBT can be used not just for developing test scripts for manual execution, but even for (refer to Figure 2) developing test scripts for automated execution. In other words, it can be used to build a test automation framework.

Figure 2: Automation Across the Test Life Cycle



Traditional Test Design

Typically, while planning test cases for software testing, testers go through software functional specification (or software requirements specification) to determine what is to be tested. [6] The SFS/SRS in turn would have been mapped to the requirements in the business requirements document. The testers then hand-craft test cases that test each specification for the software.

Test Design Using Model Based Testing

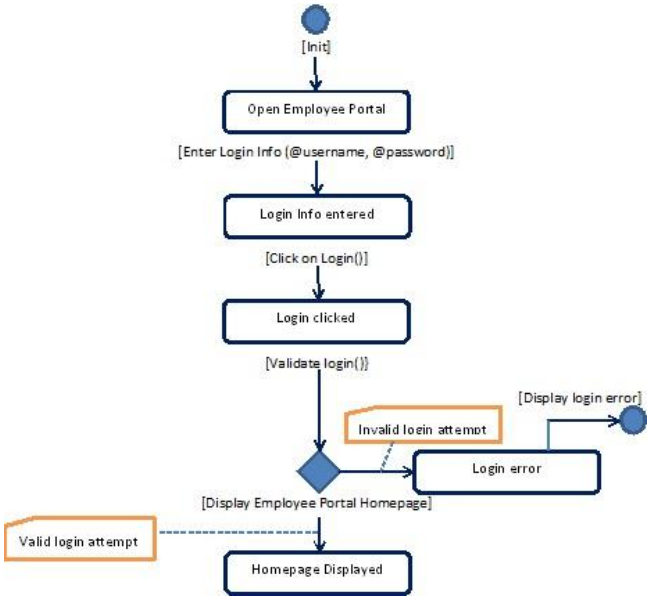
In model-based testing, we can start from either the BRD or the SRS documents by modeling the required (or specified) behavior of the software. A model is created in the form of either a finite state machine or a state chart or a Markov chain, and are commonly represented using UML. [7]

James Rumbaugh, Grady Booch and Ivar Jacobson at Rational Software created the Unified Modeling Language in 1997. It subsequently became the basis for model-based testing. [5]

A model can represent the business requirements to varying degrees, depending on the ease of representation or the focus of testing. It may be as close to the requirements as a pictorial reflection, or represent it accurately but on a smaller scale as a replica. Alternatively, due to a higher focus on key requirements, it may look like a caricature of the requirements. It may even look like a compromise due to the difficulty in representing requirements, and can be acceptable as long as the key requirements are well represented. [8]

An MBT tool is then used to parse the model and create test cases and test scripts for manual execution. [9]

Figure 3: Part of a Model Representing a WebApp's Login Functionality



MBT tools can additionally be supported using a coding language such as Java, C# or some proprietary modeling language (such as Qtronic Modeling Language [QML] used by

The SOFTWARE TEST AUTOMATION book by Mark Fewster and Dorothy Graham in 1999 is a comprehensive treatment of software test automation issues, strategies and tactics. [5]

ConformiQ) to specify test data, constraints, keywords, tags or any additional information. MBT tools support a coded model, parse the code, create test cases and even test scripts for manual execution.

Test Automation Using Model-Based Testing

Model-based testing can be taken a step farther to develop automated test scripts, too. The manual test scripts generated above can be converted into test scripts for automatic execution. Test automation through model-based testing can be considered fourth generation test automation. It supports field defect prevention, requirement-defect identification, and automatic generation of tests from models, which eliminates manual test design and reduces cost.

Model-based test automation involves three major steps. Let us go through the steps and also look at how the model in Figure 3 is processed, for testing the Web App functionality that it depicted.

A. **Creating an executable model**

This is required if the model generated was a diagrammatic one like a finite state machine. Most MBT tools help create one. If it has test data associated and necessary constraints and dependencies defined, it is an executable one. This association can be made through coding (as shown in Figure 4 below), spreadsheet or by

Choice of the right MBT tool is based on the domain of the software under test; its ability to integrate with popular test automation tools and automatically generate executable scripts is critical to building an effective test automation framewok.

other means. Then the model is detailed enough that it can be simulated.

Figure 4: Test Data Association for the Model

```
record LoginPage
{
String LoginName = "admin" prefer;
String LoginPassword = "admin" prefer;
}
```

B. **Rendering the test cases from the model to test scripts for a test execution engine**

Most MBT tools support rendering for popular test execution tools like Selenium or QTP, and unit testing frameworks such as JUnit or NUnit. The MBT tool creates an executable script with sufficient code for some of the steps, and placeholders for others (for example, where additional GUI details may be required). The resultant stub (as in Figure 5) may need further modification before it can be executed as an automated test.

Figure 5: Test Automation Stub

```
sendOpenURL( new OpenURL());
sendLoginCredential( new LoginCredential());
sendLoginButton(new LoginButton(/**BtnClicked*/ "Login"));
```

C. **Making the scripts complete and automatically executable**

The automation stubs generated from the MBT tool can then be further enhanced to make them automatically executable with test automation tools like Selenium, QTP. This can be achieved by using the record and playback

Keyword-driven testing had been coined by authors Mark Fewster and Dorothy Graham in their 1999 classic, "The SOFTWARE TEST." [5]

feature of the automation tool or manually editing the generated stubs.

There are also various open source automation frameworks available, like the Robot framework, that can be easily plugged in. These can be used to generate executable test scripts from the stubs generated and handle the concrete lower level execution of the SUT.

Frameworks integrated with MBT typically utilize the following approaches:

- **Keyword-driven** – separates test design and execution details for easier maintenance

Keyword-driven testing is a software testing technique that separates the test programming work of test automation from the actual test design. This technique works by specifying keywords for basic operations, like click button, open, close, enter, etc. These keywords are then converted into executable scripts for an automation tool.

For example, an MBT tool can produce keyword-based tests as shown below (Table 1) for a typical login page.

Sourcing the test data and directives from external sources facilitates optimization using combinatorial or pairwise techniques. This provides for high test coverage with less effort.

Table 1: Test Directives Stored External to Scripts

Window	Control	Action	Arguments
LoginPage	Browser	Open	www .loginpage.com
LoginPage		Enter	Admin, admin
LoginPage	Button	Click	Login
LoginPage		Verify	Welcome Page
LoginPage	Browser	Open	www .loginpage.com
LoginPage		Enter	Admin, password
LoginPage	Button	Click	Login
LoginPage		Verify	Error

The pseudo code in the table above represents the complete test steps. For these test cases, there is very little additional work needed that requires mapping the control and action to GUI of the SUT.

- **Data-driven** – data is persisted outside of tests in a database, spreadsheet, etc.

Data-driven testing is useful to repeat a single test with multiple test data so as to avoid having a large number of test cases, to test a similar scenario that varies only in data. It reads data from an external data source instead of using the same hard-coded values each time the test runs. MBT integrates with the data-driven approach by associating test data with the different states, transitions or actions. It is especially useful where SUT needs to be tested with a large amount of input values.

Table 2 shown below is an example of a data table generated for a login screen to test the same login

functionality with multiple data sets, both positive and negative.

Table 2: Test Data Stored External to Scripts

UserName	Password	Valid
Admin	Password	True
Admin	p@ssword	False
superuser	India@123	True
Superuser	India123	False
Nouser	Password	False

This data set is then read by the test script generated to create multiple test cases from each test scenario. If the test data is huge, it is advisable to optimize the test data combinations using combinatorial testing or pair-wise testing. MBT, when combined with these optimization techniques provides higher testing coverage with less effort, low maintenance and is very adaptive to changes.

- **Structured** – uses control structures like 'if-else', 'switch', 'for', 'while' statements

This approach enables a structured and modular view of the entire system functionality. It helps represent the functionality of the sytem using functions and procedures. These functions are then used in a hierarchical fashion to construct larger tests, realizing a particular test case. This insulates the tests from

IBM launched its automated test script generator, IBM Rational Functional Tester, in 2002. It had originally been released as RobotJ. [5]

modifications in the SUT and provides modularity in the test design.

- **Hybrid** – The combination of two or more of the patterns used

The most commonly implemented approach is a combination of the above, using their strengths and trying to mitigate their weaknesses. This hybrid test automation framework is what most frameworks evolve into over a period of time and multiple projects.

For example, the keyword and data-driven approaches, when applied to a login page example given above, will simplify as Table 3 below:

Table 3: Test Data and Directives Stored External to Scripts

Window	Control	Action	Arguments
LoginPage	Browser	Open	www .loginpage.com
LoginPage		Enter	@uname, @pwd
LoginPage	Button	Click	Login
LoginPage		Verify	@valid

Many of the commercially available MBT tools have built-in integration with test automation tools to generate complete test automation scripts (such as the one in Figure 6) directly from models. This makes testers' lives much easier and greatly reduces testing cost.

Figure 6: Final Test Automation Script

```
public void sendOpenURL(OpenURL argOpenURL)
        throws TestSuiteException {
try {
        errMessage = “problem while opening the URL”;
        baseUrl =
objectsInstanceService.getApplicationProPropertyInstance().getP
roperty(“TA_URL”);
        Log.warn(“ --> Connecting server : “ + baseUrl);
        Log.info(“ --> URL entered :” + baseUrl);
} catch (TimeoutException e) {
        handleExceptionforFailure(e, errMessage);
} catch (NoSuchElementException e) {
        handleExceptionforFailure(e, errMessage);
} catch (SeleniumException e) {
        handleExceptionforFailure(e, errMessage);
} catch (Exception e) {
        handleExceptionforFailure(e, errMessage);
        }
}
```

```
public void sendLoginCredential(Login Credential
argLoginCredential)
        throws TestSuiteException {
try {
        errMessage = “problem while entering the credentials”;
        String userName =
ObjectsInstanceService.getApplicationProPropertyInstance().getP
roperty(“TA_userName”);
        String password =
ObjectsInstanceService.getApplicationProPropertyInstance().getP
roperty(“TA_password”);
// enter username
        driver.findElement(By.id(“InUserName”)).clear();
        driver.findElement(By.id(“InUserName”)).sendkeys(user
name);
// enter password
        driver.findElement(By.id(“InPassword”)).clear();
        driver.findElement(By.id(“InPassword”)).sendkeys(pass
word);
        log.debug(“ --> ID of Passoword : “ + ”InPassword”);
        Log.info(“ --> Login credential entered);
        Log.info(“ --> Username: “ + username);
        Log.info(“ --> Passowrd: “ + password);
        If(ScreenshotMode.equals(“ALL”)) {
        CommonTasks.getScreenShots(file,testCaseName,
ObjectsInstanceService.getApplicationProPropertyInstance().getP
roperty(“TA_ScreenShot”),i);
        I++;
        }
} catch (TimeoutException e) {
        handleExceptionforFailure(e, errMessage);
} catch (NoSuchElementException e) {
        handleExceptionforFailure(e, errMessage);
} catch (SeleniumException e) {
        handleExceptionforFailure(e, errMessage);
} catch (Exception e) {
        handleExceptionforFailure(e, errMessage);
        }
}
```

Model-based testing helps automate the creation of test cases and test scripts (manual and automated), providing for complete coverage and easy maintenance in a cost-efficient way.

Benefits

The model-based test automation approach combines the benefits of model-based testing with that of modularity frameworks for automated test script design.

Benefits of Model-Based Testing:

- **Greater coverage**: Allows complete and exhaustive test coverage as we depend on machine/tool for parsing requirements that have already been captured as a model.
- **Shorter cycle time**: Allows accelerated test case creation once the model is made available. Reduces time and effort involved in creating the test cases.
- **Easy Maintenance**: Facilitates easier maintenance of test cases and test scripts. Models which are easier to understand can quickly be modified to regenerate test cases.

Benefits of modularity frameworks for automated test script design:

- **Faster Test Automation**: Quick conversion of manual test cases to automated test scripts. Reduces time and effort involved in coding test scripts.
- **Eliminates the necessity to modify code**: Elimination of human error that can creep in while modifying large test scripts.

Model-based testing can be considered Fourth Generation test automation.

It is a step beyond record-and-playback (First Gen), scripted automation (Second Gen) and data/keyword-driven testing (Third Gen).[10]

Model-based test automation supports field defect prevention, requirement-defect identification, and automatic generation of tests from models, which eliminates manual test design and reduces cost.

Conclusion

A test automation framework can be built around model-based testing that begins with automating the test design exercise and the development of manual test procedures. It further uses automation frameworks to attain modularity in automated test script design, and integrates with test execution tools for automated test execution.

Such a framework will help test organizations to:

- Greatly scale when test desingers and test automation developers are in short supply by training regular testers in model-based testing and test automation
- Deliver test automation quickly so that regression cycles can be automated right from the first cycle, and save effort and cost that would otherwise have been expended on manual testing
- Derive the benefits of model-based testing viz. exhaustive voverage and easier maintenance, for automated testing, too.

References

1. 2011 Model-Based Testing User Survey: Results & Analysis by Robert V. Binder http://www.robertvbinder.com/docs/arts/MBT-User-Survey.pdf
2. Data Driven and Keyword Driven Test Automation Frameworks by Pekka Laukkanen http://eliga.fi/Thesis-Pekka-Laukkanen.pdf
3. Wikipedia: Test Automation http://en.wikipedia.org/wiki/Test_automation
4. NASSCOM Research & Intelligence - Software Testing: Shifting from Functional to Business Assurance http://survey.nasscom.in/sites/default/files/researchreports/softcopy/Software%20Testing%20Report_Secured.pdf
5. The History of Software Testing by Joris Meerts http://www.testingreferences.com/testinghistory.php
6. Wikipedia: Requirements Analysis https://en.wikipedia.org/wiki/Requirements_analysis
7. MBT: A superior alternative to traditional software testing - HCL Blog by Ambica Jain on 11 Oct 2012 http://www.hcltech.com/blogs/engineering-and-rd-services/model-based-testing-%E2%80%93-superior-alternative-traditional-software-te
8. Modeling Introduction by Hani Achkar http://testoptimal.com/ref/TO_Presentation/Modeling_101.pdf
9. Model-Based Testing (MBT) – HCL Whitepaper by Naveen Jain published on 14 Jun 2013 http://www.hcltech.com/white-papers/engineering-services/model-based-testing
10. Generations of Test Automation Tools by Mark Utting http://www.cs.waikato.ac.nz/research/mbt/lectures/TestAutomation.ppt

Author Info

Johnson Selwyn has led large development and test teams in both the embedded and application software spaces. Currently, he is part of the Practice Group, and is responsible for conceptualizing differentiated testing services enabled by solution accelerators.

Ambica Jain has experience working in various technologies and domains. At present, she is part of the Practice Group, and her role involves identifying ways to make life easier for testers by automating testing to the maximum possible and designing frameworks to achieve it most efficiently.

Hello, I'm from HCL's Engineering and R&D Services. We enable technology led organizations to go to market with innovative products and solutions. We partner with our customers in building world class products and creating associated solution delivery ecosystems to help bring market leadership. We develop engineering products, solutions and platforms across Aerospace and Defense, Automotive, Consumer Electronics, Software, Online, Industrial Manufacturing, Medical Devices, Networking & Telecom, Office Automation, Semiconductor and Servers & Storage for our customers.

For more details contact**: ers.info@hcl.com**
Follow us on twitter: http://twitter.com/hclers
Our blog: http://www.hcltech.com/blogs/engineering-and-rd-services
Visit our website: http://www.hcltech.com/engineering-rd-services

HCL