International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 1 Issue 4, June - 2012

Automated Model Based Testing

[#1] Jyoti Mishra, [#2] Irphan Ali, [#3]Anubhaw Kumar Upadhyay
[#1] Research Scholars, NIET Greater Noida
[#2] Assistant Professor, NIET Greater Noida
[#3] Research Scholars, Tula's Institute Dehradun

Abstract

Software testing is becoming more and more difficult task every day because in the current software engineering cycle the design and testing activities are separated, which leads to a situation where test cases are not in harmony with the actual application. Traditional testing techniques may not always be suitable for adequate, thorough, and extensible testing of critical and complex software in a resource and time constrained software development environment. One way to solve this problem is to take a model of the application into use, which can be, e.g., a user interface model and from which the test cases can be derived automatically. This technique is known as model based testing. The model can also be used for, e.g., automatic code generation. Model-based testing (MBT) is an evolving technique for generating test cases automatically from a behavioral model of a system under test.

The main objectives of this paper were to deploy the model based approach into an organization, to develop an automated model based testing tool in order to test the approach and to show proof of financial benefits of the approach as compared to the Traditional testing technique. In this paper we first thoroughly explore the model based Testing technique and then deploy it in an Organization and then comparison between traditional testing methods and automated model based testing is conducted.

1 Traditional Software Testing

Traditional software testing consists of the tester studying the software system and then writing and executing individual test scenarios that exercise the system. These scenarios are individually crafted and then can be executed either manually or by some form of capture/playback test tool.

This method of creating and running tests faces at least two large challenges:

First, these traditional tests will suffer badly from the "pesticide paradox" (Beizer, 1990) in which tests become less and less useful at catching bugs, because the bugs they were intended to catch have been caught and fixed.

Second, handcrafted test scenarios are static and difficult to change, but the software under test is dynamically evolving as functions are added and changed. When new features change the appearance and behavior of the existing software, the tests must be modified to fit. If it is difficult to update the tests, it will be hard to justify the test maintenance costs.

Model-based testing alleviates these challenges by generating tests from explicit descriptions of the application. It is easier, therefore, to generate and maintain useful, flexible tests.

2 Test Automation

Organizations often seek to reduce the cost of testing. Most organizations aren't comfortable with reducing the amount of testing so instead they look at improving the efficiency of testing. Luckily, there are a number of software vendors who claim to be able to do just this! They offer automated tools which take a test case, automate it and run it against a software target repeatedly. Music to management ears! However, there are some myths about automated test tools that need to be dispelled:

2.1 Automated testing does not find more bugs than manual testing – an experienced manual tester who is familiar with the system will find more new defects than a suite of automated tests.

2.2 Automation does not fix the development process – as harsh as it sounds, testers don't create defects, and developers do. Automated testing does not improve the development process although it might highlight some of the issues.

2.3 Automated testing is not necessarily faster-The upfront effort of automating a test is much higher than conducting a manual test, so it will take longer and cost more to test the first time around. Automation only pays off over time. It will also cost more to maintain.

2.4 Everything does not need to be automated-some things don't lend themselves to automation, some systems change too fast for automation, some tests benefit from partial automation – you need to be selective about what you automate to reap the benefits. But, in their place, automated test tools can be extremely successful.

3 Need of Automated Testing

Automated Testing is automating the manual testing process currently in use. This requires that a formalized "manual testing process" currently exists in the company or organization. Minimally, such a process includes:

- ➢ Detailed test cases, including predictable "expected results", which have been developed from Business Functional Specifications and Design documentation
- ➢ A standalone Test Environment, including a Test Database that is restorable to a known constant, such that the test cases are able to be repeated each time there are modifications made to the application

The real use and purpose of automated test tools is to automate regression testing. To perform it there should be a database of detailed test cases that are repeatable, and this suite of tests must be run every time there is a change to the application to ensure that the change does not produce unintended consequences.

An automated test script is a program. Automated script development, to be effective, must be subject to the same rules and standards that are applied to software development.

According to Customer request, the needed automated testing tool should provide ability to perform the following types of testing:

- ➢ Regression Testing (for Web interfaces) - not just navigation and links, expected and actual results
- ➢ Load/Stress/Volume Testing
- ➢ Unit/Integration Testing

There are also some additional features of the automated testing tool that should be taken into consideration:

- ➢ Cost
- ➢ Inter-operability – ability to work together with other tools
- ➢ Capacity - particularly on load testing, how many users at once, virtual clients, multiple physical client spanning etc

4 Automated Testing Tools

There are a number of Automated Testing tools on the market today. The following is just a representative list of the major vendor products that are most widely used.

- Android
- Atesto Functional Testing Service
- AutoTester for Windows
- AutoTester for OS/2
- CitraTest
- e-Monitor
- e-Tester
- eValid
- SilkTest
- Smalltalk Test Mentor
- SQA TeamTest: ERP Extension for SAP
- TestBench400
- SQA TestFoundation for PeopleSoft
- Tasker
- Test Now
- TestQuest Pro Test Automation System
- TestRunner
- Unified TestPro (UTP)
- Vermont High Test Plus
- Visual Test
- WebKing
- WinRunner
- xrc - X Remote Control
- XRunner

5 Automated Models Based Testing

Model based testing can be summarized in one sentence; it is essentially a technique for automatic creation of test cases from specified software model. The key advantage of this technique is that the test generation can systematically derive all combination of tests associated with the requirements represented in the model to automate both the test design and test execution process [5].

Generating test cases from a finite state machine is usually straightforward and easy task. The states in the machine contain information about what the state is; in the case of actions, they describe the action that is executed, and in the case of responses, they describe formally, what the response is. If the test cases are to be executed manually, this is all the information that is needed. If the goal is to generate automated test cases, the states also need to contain all the necessary test automation parameters that are needed by the test automation system used. The test automation parameters can of course be derived also from some other information, e.g. by using a text parser to generate the parameters from the verbal descriptions automatically.

As the states contain all the necessary information, actual test case generation is simple. The simplest approach is to traverse randomly through the state transition diagram, from start to the end. One test case is constituted of the states visited during traversing. Frequently it is wanted that the transition coverage criterion is 100% and then random traversing is out of the question and a more sophisticated algorithm is needed. In [4], some graph traversal algorithms are presented that aim to meet the criterion, and the challenge is to choose the most effective one.

5.1 Benefits of Automated Model Based Testing

Many studies conducted have shown that model based testing is effective, especially when used to test small applications, embedded systems, user interfaces and state-rich systems with reasonably complex data. Rosaria and Robinson (2000) studied testing graphical user interfaces, Agrawal and Whittaker (1993) embedded control software and Avritzer and Larson (1993) phone systems.

Usually the most attractive attribute of model based testing is thought to be the automatic generation of test cases, but that is not all. Model of software can help refining unclear and poorly defined requirements. By eliminating model defects before the coding begins and automating the test case creation the result is significant cost savings and higher quality code. Figure 6 shows the differences between current defect discovery and elimination process (marked “Old”) and early defect discovery (marked “New”).

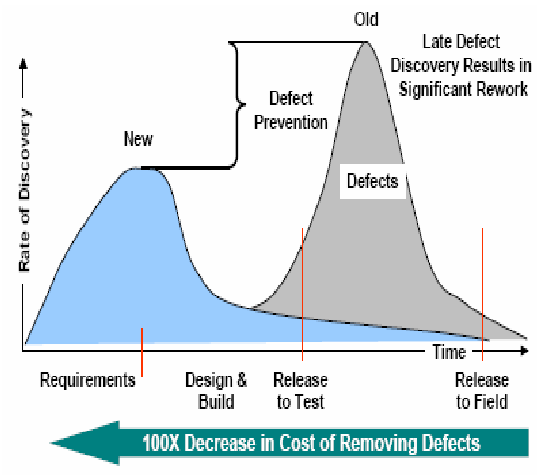

Figure 1: Savings caused by earlier defect discovery

The figure shows that the sooner the defects are detected and fixed, the less the costs of fixing them will be. Other benefits that are more related to testing include e.g. the following.

Comprehensive tests: if the model is a complete abstraction of the software, it is possible to automatically create test cases which cover every possible transition of the model by using graph algorithms.

Defect discovery: model based test automation discovers defects more effectively than manual methods. The article demonstrated this with a case study in which manual method uncovered 33 defects in a system, and model based method all of those and in addition 56 more.

As it was shown in this section the benefits of model based testing are huge if modeling and all the related tasks are done efficiently, but it also has some difficulties and drawbacks.

5.2 Difficulties and Drawbacks of Automated Model Based Testing

Almost every research on model based testing agrees on one thing: deployment of model based testing into an organization requires significant efforts and investments. In [4], the following three reasons for the needed efforts and investments are presented:

Excessive amount of skills is required from the testers. They need to be familiar with the model, which means knowledge of different forms of state machines, formal languages, and automata theory. In addition, expertise in tools and scripts is required when test automation is going to be used.

A large initial effort in terms of man-hours is required. The type of the model has to be carefully selected, different parts of software have to be divided so that the modeling is easier because of the smaller areas and the actual model has to be built.

6 Conclusions

Usually the most attractive attribute of model based testing is thought to be the automatic generation of test cases, but that is not all. Model of software can help refining unclear and poorly defined requirements. By eliminating model defects before the coding begins and automating the test case creation the result is significant cost savings and higher quality code.

On the basis of above study we can say that model based testing is effective, especially when used to test small applications, embedded systems, user interfaces and state-rich systems with reasonably complex data.

References

The result of the comparison shows that the automated model based testing techniques have benefits over traditional technique of testing. The main benefits are in fields of time and money.

[1] Beizer, B. Software Testing Techniques. 2nd Ed. New York, USA: Van Nostrand Reinhold Co, 1990. 550 p. ISBN 0-442-20672-0.

[2] El-Far, I. K. & Whittaker, J. A. Model-based Software Testing. In:Marciniak, J. (ed.), Encyclopedia on Software Engineering, Volume 1. New York, USA: John Wiley & Sons Inc, 2001. pp. 825-837. ISBN 0-471-21008-0.

[3] Robinson, H. Finite state model-based testing on a shoestring.International Conference on Software Testing Analysis and Review, San Jose, California, USA, 1999.

[4] Robinson, H. Graph Theory Techniques in Model-Based Testing. 16th International Conference and Exposition on Testing Computer Software, Los Angeles, California, USA, 1999.

[5] Blackburn, M., Busser, R. & Nauman, A. Why Model-Based Test Automation is Different and What You Should Know to Get Started. International Conference on Practical Software Quality and Testing, Washington, USA, 2004.