MODEL-BASED TESTING FOR EARLY FAULT DETECTION
Amandeep Singh Dhillon

Abstract - Model-based testing (MBT) is about testing a software system by using a model of its behaviour. To benefit fully from MBT, automation support is required. This paper presents a systematic review of prominent MBT tool support where we focus on tools that rely on state-based models. The systematic review protocol precisely describes the scope of the search and the steps involved in tool selection. Precisely defined criteria are used to compare selected tools and comprise support for test coverage criteria, level of automation for various testing activities, and support for the construction of test scaffolding. The results of this review should be of interest to a wide range of stakeholders: software companies interested in selecting the most appropriate MBT tool for their needs; organizations willing to invest into creating MBT tool support; researchers interested in setting research directions.

Keywords. state-based testing, transition-based testing, systematic review

1 INTRODUCTION

Software testing requires the use of a model to guide such efforts as test selection and test verification. Often, such models are implicit, existing only in the head of a human tester, applying test inputs in an ad hoc fashion. The mental model testers build encapsulates application behavior, allowing testers to understand the application’s capabilities and more effectively test its range of possible behaviors.

There is an abundance of testing styles in the discipline of software engineering today. Over the last few decades, many of these have come to be used and adopted by the industry as solutions to address the increasing demand for assuring software quality. During the last ten odd years, perhaps as an outcome of the popularization of object orientation and models in software engineering, there has been a growth in black box testing techniques that are collectively dubbed model-based testing.

The cost of finding and fixing faults in software typically rises as the development project progresses into a new phase. Faults that are found after the system has been delivered to the customer are many times more expensive to track down and correct than if found during an earlier phase. The difference in cost of faults between development phases is dependent on the development method used. It is claimed that projects that use agile development practices have a fault cost curve that is less steep. Current and future trends for software include increasingly complex requirements on interaction between systems. For systems with a high degree of interaction, many defects cannot be detected on the unit or component level. Instead, the importance of integration and system-level testing increases. Such systems cannot avoid having a steep fault cost curve, even if agile development practices are used. Furthermore, the increased complexity means that a system may have potentially infinite combinations of inputs and resulting outputs. It is difficult to get satisfactory coverage of such a system with hand-crafted manual or automatic test cases.

Model based testing is a test technique where test cases are generated from a model of the system. There are model-based testing tools that can automate the generation of test cases from a behavioral model, including test oracles that can determine whether the system under test behaved correctly at the execution of the test case. Test cases generated from a model have been shown to give a high coverage of system interaction points, given that the generation is carefully guided.

Model-based Testing

Simply put, a model of software is a depiction of its behavior. Behavior can be described in terms of the input sequences accepted by the system, the actions, conditions, and output logic, or the flow of data through the application’s modules and routines. In order for a model to be useful for groups of testers and for multiple testing tasks, it needs to be taken out of the mind of those who understand what the software is supposed to accomplish and written down in an easily understandable form. It is also generally preferable that a model be as formal as it is practical. With these properties, the model becomes a shareable, reusable, precise description of the system under test.

We cannot possibly talk in detail about all software models. Instead, we introduce a subset of models that have been useful for testing and point to some references for further reading. Examples of these are finite state machines, state charts, the unified modeling language (UML) and Markov chains. Studies of software process improvement suggest that regardless of whether a quality initiative is technical or organizational, the human factor should be considered, because of the potential barriers to change. Model based testing is not just a matter of generating tests, and executing them to detect defects. It involves several other activities, such as creation of the system model, analyzing the output, reporting defects and generating reports. Another important activity is regression testing, which is often cited as the most important benefit of test automation. The model-based testing practice must therefore be integrated into the project’s test process. This integration can meet resistance as existing local practices.
may directly conflict with the model-based testing technique.

2. Related Research

The term MBT is currently used for a wide variety of test generation techniques. The four main approaches known as MBT are described by Utting and Legeard [32]:

1. Generation of test input data from a domain model
2. Generation of test cases from an environment model
3. Generation of test cases with oracles from a behavior model
4. Generation of test scripts from abstract tests.

MBT automates the detailed design of test cases and the generation of the traceability matrix, which measures the coverage of requirements for each test case. Instead of writing hundreds of test cases, the test designer constructs an abstract model of the system under test. The MBT tool is used to generate a set of test cases from that model. As well as having the advantage of reducing design time, a variety of test suites can also be generated from the same model simply by using different test selection criteria.

Generating test cases from high-level specifications is not a recent idea. In 1986 Hayes showed how to systematically derive tests of abstract data structures from a formal specification. At that time, however, the generation and execution of test cases was performed manually. Today, there is a growing number of tools available that automate many of the steps involved in model-based testing. Utting and Legeard provide a comprehensive overview of model-based testing tools. Quick Check, developed by Quviq AB, is a testing tool for guided random and model-based testing. QuickCheck can simplify a failed test case to a minimal failing test case, thereby reducing the problem of deducing the cause of failure for complex test cases. A minimal test case is a test case where every part of the system input is significant in reproducing the failure.

QuickCheck provides a framework for modelling the system under test using an Abstract StateMachine. A model is built using Erlang, a general-purpose programming language, with support of the library provided with Quick Check.

3. METHODS

The study was conducted using the action research method. This was motivated by the practice oriented nature of the study, and the author’s involvement in both practice and research. Action research is cyclic. Each cycle typically includes planning, acting, observing, and reflecting. The studied development project used a development process based upon the DSDM framework (The DSDM framework is further discussed in DSDM, Business Focused Development). The length of each time-box was approximately four weeks. The customer conducted an acceptance test on each system release. Prior to this study, model-based testing was used by some developers for testing on the unit level. The tool used was Quick Check.

The authors role in this study, except from data collection and analysis, was to introduce model-based testing as a technique for system testing in the project. The author also constructed the model of the system, and executed the tests generated from the model. The risk of bias inherent due to the authors involvement and interventions is acknowledged.

4. DATA COLLECTION AND ANALYSIS

Both quantitative and qualitative data was collected in this study. Found faults were analyzed and the fault-slip-through to acceptance testing measured. The impact of the model-based testing was verified through qualitative interviews with the test-manager, developers and a customer representative. In addition, the experiences of system-level modeling, test execution, and reporting of test results were logged.

FAULT ANALYSIS

Faults found during system testing and acceptance testing were measured at the end of each time-box for each of the two sub-systems. Data on found faults was collected from the organization issue tracking system. Both the internal organization and the customer reports detected anomalies into this system. The reports include the details of the anomaly, the reporter and the date of the report. This data was sufficient for the FST measurement. Each anomaly report was analyzed according to the following criteria:

1. The anomaly has been confirmed to have been caused by a fault in one of the two sub-systems
2. The fault related to a functional aspect of the system.

5 RESULTS

Detected faults

The following sections presents the faults found in the studied releases. We present faults detected during system testing and acceptance testing of two releases of the messaging gateway, Release X and Release X+1. We also present the faults detected in the interim release, Release X+0.5 that only included changes in the E-mail gateway.

A high number of faults were found in both subsystems during the acceptance test of release X. Over half of these defects should have been found during system testing. We can thus not measure any significant difference in fault-slip-through between the two subsystems for this release. A high number of faults were detected by the model-based tests during system testing of Release X+0.5 and Release X+1. Of the faults detected in the E-mail gateway during the acceptance testing of release X+1, most could not have been detected on previous test levels. There is a significant difference compared to the IM gateway, where most of the faults detected during acceptance testing should have been found earlier. The finding that our improvement initiative did not see effect until Release X+1 is in line with prior studies that suggest that successful implementation of improvements requires multiple iterations.

6 CONCLUSIONS AND FUTURE WORK

This study set out to contribute to the understanding of system-level model based testing as a test technique for early fault detection. Our experiences of modeling and test execution are generally in line with those reported by prior studies. We contribute further to the understanding of system-level. The test results for the two subsystems shows that, for the subsystem tested with the model-based tests, significantly less faults slipped through from system testing to customer acceptance testing.
This supports our hypothesis H1, that model-based testing would decrease our fault-slip-through from system testing to customer acceptance testing. During the study, we observed that the customers’ confidence in the system is dependent on the availability of test reports. The customer (external organization) requires these reports to see that the system has been sufficiently tested. As identified by prior studies, it is also important to make the results of the model-based testing visible within the internal organization, to get commitment to the technique.

REFERENCES