(IJRST) 2012, Vol. No. 2, Issue No. II, Apr-Jun **ISSN: 2249-0604**

"MODEL-BASED SOFTWARE TESTING"

VIVEK KUMAR

ABSTRACT

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. When these mental models are written down, they become sharable, reusable testing artifacts. In this case, testers are performing what has become to be known as model-based testing. Model-based testing has recently gained attention with the popularization of models in software design and development. There are a number of models of software in use today, a few of which make good models for testing.

Keywords-*Software behavior models, Finite state machines, Test case generation*

INTRODUCTION

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. We open this discussion of model-based testing with a few definitions of common terms. Some of the more popular software models are explored, and some guidelines for selecting models are presented.

MODEL-BASED TESTING

Models

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. There are numerous such models, and each describes different aspects of software behavior.

Models in Software Testing
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.

3 UNDERSTANDING THE SYSTEM UNDER TEST

The requirement common to most styles of testing is a well-developed understanding of what the software accomplishes. Forming a mental representation of the system functionality is a prerequisite to building models. This is a nontrivial task as most systems today typically have convoluted interfaces and complex functionality. Moreover, software is deployed within gigantic operating systems among a clutter of other applications, dynamically linked libraries, and file systems all interacting with and/or affecting it in some manner. To develop an understanding of an application, therefore, testers need to learn about both the software and its environment.

- Identify the users of the system. Each entity that either supplies or consumes system data, or affects the system in some manner needs to be noted. Consider user interfaces keyboard and mouse input the operating system kernel.

Building the Model
In general, state model-based testers define high-level state abstractions and then refine these abstractions into an actual state space. Enumerating the state space by hand is formidable except when modeling some small subsystems. State abstractions are based on inputs and information about the applicability of each input and the behavior that the input elicits. Indeed, the standard definition of a state used in various methodologies.

Running the Tests
Although, tests can be run as soon as they are created, it is typical that tests are run after a complete suite that meets certain adequacy criteria is generated. First, test scripts are written to simulate the application of inputs by their respective users. Next, the test suite can be easily translated into an executable test script. Alternatively, we can have the test generator produce the test script directly by annotating the arcs with simulation procedures calls.

ORACLES AND AUTOMATION

Automation and oracles make interesting bed fellows. On the one hand, oracles are crucial to making automation work. What good is it to execute one million tests cases if we cannot tell which ones passed and which ones failed, On the other hand, automation itself makes writing an oracle even more difficult Because manual testing is slow, fewer tests can be performed and oracles do not have to be as comprehensive. Indeed, they must cover only those behaviors that the manual tester has time to perform. Moreover, oracles for manual testing can also be manual because there is usually time to verify screen output during the slow process of manual test execution.

CONCLUSION

Model-based testing, in all its simplicity, appears to be a useful and efficient testing method for quickly reaching large test coverage in a system, without enormous testing costs. Because model-based testing implies radical changes in the software design and testing processes, a full scale adoption of the technique requires thorough evaluation of its benefits and drawbacks, before seriously attempting to take it into use. Even though radical benefits and savings and even ten-fold productivity improvements have been reported in the academics. As long as modeling is performed one way or another in software projects, model-based testing will continue to have a certain appeal, simply because of the reuse value provided by existing design models. Another appeal of model-based testing is its flexibility, allowing a vast number of tests, with different objectives and levels of coverage, to be developed with low cost and effort, once the model is ready.

REFERENCES

1. Apfelbaum, L. and Doyle, J., Model based testing. Proceedings of the 10th International Software Quality Week, May 1997.
2. Binder, R. V., Testing object-oriented systems: models, patterns, and tools. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.
3. El-Far, I. K. and Whittaker, J. A., Model-based software testing. Encyclopedia of Software Engineering, Marciniak, J. J., editor. John Wiley & Sons, Inc., 2001.
4. Hartman, A., Adaptation of model based testing to industry (presentation slides). Agile and Automated Testing Seminar, Tampere University of Technology, Tampere, Finland, August 2006.
5. Rosaria, S. and Robinson, H., Applying models in your testing process. Information and Software Technology, 42,12(2000), pages 815–824.
6. Veanes, M., Campbell, C., Schulte, W. and Kohli, P., On-the-fly testing of reactive systems. Technical Report MSR-TR-2005-05, Microsoft Research, January 2005.
7. Zhen R. D., Model-driven testing with UML 2.0. Proceedings of Second European Workshop on Model Driven Architecture, Akehurst, D. H., editor, University of Kent at Canterbury, United Kingdom, August 2004.