Volume 2, No. 4, April 2011
Journal of Global Research in Computer Science
ISSN-2229-371X
RESEARCH PAPER
Available Online at www.jgrcs.info

MODEL BASED TESTING CONSIDERING STEPS, LEVELS, TOOLS & STANDARDS OF SOFTWARE QUALITY

[1]Sanjeev Dhawan, [2]Nirmal Kumar and [3]Shiva Saini

Department of Computer Engineering, UIET, Kurukshetra University, Kurukshetra.

nirmal.sirohi@gmail.com

Abstract **-** This paper gives an overview to the topic of model-based testing. The model based testing process is described, and the steps available at each stage are considered. The different types of tool necessary to support the process are explained and example tools listed along with the standards they support. The position for standards in model-based testing is examined, and the new skills required by tester are discussed. Throughout research on model-based testing in the last 5-10 years has verified the probability of this approach. It has been shown that it can be cost-effective, and has developed a variety of test generation strategies and model coverage criteria. Some commercial tools have started to emerge, from the USA (T-Vec, Reactive Systems, I-logix), and also from Europe (Conformiq, Leirios Technologies, Telelogic), as well as a wide variety of academic and research tools [BFS05]. The discussion in this paper is limited to functional testing, because model-based testing is less mature in other areas. Finally, means of determining the appropriateness of projects for model-based testing are considered. In this paper following factors are analyzed: Model Based Testing, Steps, Levels, Methods & tools that affect MBT.

INTRODUCTION

We all use models to make testing – otherwise we wouldn't have a hint whether a test will be passes or fails – these models allow us to know how the software should perform in a given condition. However, most people's models are very personal and never see the light of day, only existing for a brief time in the tester's head. With model-based (or model-driven) testing, the model of the system's behaviour is completed explicit and used as the basis for the whole automation of the testing (see figure 1).

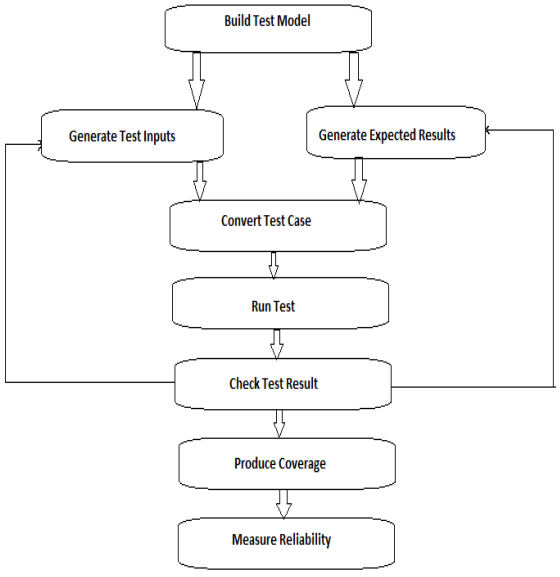

Figure 1: Model Driven Testing

The Model Based Testing Process

Model-based testing refers to the processes and techniques for the automatic beginning of abstract test cases from abstract formal models, the generation of existing tests from abstract tests, and the manual or automated execution of the resulting existing test cases. Therefore, the key points of model-based testing are the modelling principles for test generation, the test generation strategies and techniques, and the concretization of conceptual tests into existing, executable tests. [1]

MODEL BASED TESTING STEPS

1. **Design a Test Model.** The model generally called the test model represents the expected behaviour of the system under test (SUT). Standard modelling languages such as UML is used to formalize the control points and observation points of the system, the expected dynamic behaviour of the system, the business entities associated with the test, and some data for the initial test configuration. Model elements such as transitions or decisions are linked to the requirements, in order to ensure bi-directional traceability between the requirements and the model, and later to the generated test cases. Models must be precise and complete enough to allow automated derivation of tests from these models.

2. **Select some Test Generation Criteria.** There are usually an infinite number of possible tests that could be generated from a model, so the test analyst chooses some Test Generation Criteria to select the highest priority tests, or to ensure good coverage of the system behaviour. One common kind of test generation criteria is based on structural model coverage, using well known test design strategies such as equivalence partitioning, cause-effect testing, pair-wise testing, process cycle coverage, or boundary value analysis (see [1] for more details on these strategies). Another useful kind of test

generation criteria ensures that the generated test cases cover all the requirements, perhaps with more tests generated for requirements that have a higher level of risk. In this way, model-based testing can be used to implement a requirement and risk-based testing approach. For example, for a non-critical application, the test analyst may choose to generate just one test for each of the nominal behaviours in the model and each of the main error cases; but for one of the more critical requirements, she/he could apply more demanding coverage criteria such as all loop-free paths, to ensure that the businesses processes associated with that part of the test model are more thoroughly tested.

3. **Generate the tests.** This is a fully automated process that generates the required number of (abstract) test cases from the test model. Each generated abstract test case is typically a sequence of high-level SUT actions, with input parameters and expected output values for 2 each action. These generated test sequences are similar to the high-level test sequences that would be designed manually in action-word testing [2]. They are easily understand by humans and are complete enough to be directly executed on the SUT by a manual tester. The test model allows computing the expected results and the input parameters. Data table may be use to link some abstract value from the model with some concrete test value. To construct them executable using a test automation tool, a further concretization phase automatically translates all abstract test case into a concrete (executable) scripts, using a user-defined mapping from abstract data values to concrete SUT values, and a mapping from abstract operations into SUT GUI actions or API calls. For example, if the test execution is via the GUI of the SUT, then the action words are linked to the graphical object map, using a test robot such as **HP Quick Test Professional**, IBM Rational Functional Tester or the open-source robot Selenium. If the test execution of the SUT is API-based, then the action words need to be implemented on this API. This can be a direct mapping or a more complex automation layer. The expected results part of each abstract test case is translated into oracle code that will check the SUT outputs and decide on a test pass/fail verdict. The tests generated from the test model may be structured into multiple test suites, and published into standard test management tools such as Quality Center, IBM Rational Quality Manager or the open-source tool TestLink. Maintenance of the test repository is done by updating the test model, then automatically regenerating and republishing the test suites into the test management tools;

4. **Execute the Tests.** The generated concrete tests are typically executed either manually or within a standard automated test execution environment, such as HP QuickTest Professional or IBM Rational Functional Tester. Either way, the result is that the tests are executed on the SUT, and we find that some tests pass and some tests fail. The failing tests indicate a discrepancy between the SUT and the expected results designed in the test model, which then needs to be investigated to decide whether the failure is caused by a bug in the SUT, or by an error in the model and/or the requirements. Experience shows that model-based testing is good at finding SUT errors, but is also highly effective at exposing requirements errors [even far before executing a single test (thanks to the modelling phase). [4]

III. LEVELS, TOOLS AND STANDARDS FOR MODEL BASED TESTING

LEVELS OF TESTING

During development and maintenance life cycles, test cases can be applied to very small units, collection of units, or whole systems. Model-based testing can support test activities at all levels. At the lowest level, model-based testing can be used to apply a single software module. By modeling the input parameters of the module, a small but rich set of tests can be developed rapidly. This approach can be used to assist develo1pers during unit test activities. An intermediate-level application of model-based testing is used to checking simple behaviors; this is what we call a single step in an application. Examples of a single step are performing an addition operation, inserting a row in a table, sending a message, or filling out a screen and submitting the contents. Generating tests for a single step requires simply one input data model, and allows computation of the expected outputs without creating prediction that is more complex than the system under test. A big challenge that offers comparably better benefits is using model-based testing at the level of complex system behaviors (sometimes known as flow testing). Step-oriented tests can be used to generate comprehensive test suites. This type of testing mostly represents customer usage of software. At our work, we have selected sequences of steps based on operational profiles [6] and used for the combinatorial test generation approach to choose values tested in each step. An alternate approach to flow testing uses models of a system's behavior instead of its inputs to generate tests. [5]

TOOLS & STANDARDS

The tools used to create the models used in model-based testing are normally very similar to those uses by developers, and could be identical in some cases. Where hybrid notations are used then open source development tools can be reasonably easy to modify, even though in the future it is expected commercial development tool vendors will provide the essential added functionality for testers. The tools for test case generation are particular to the model notation used and the test coverage criteria to be achieved. These tools are not currently used in traditional software testing or development and they are specific to model-based testing. Because of their belief on the input model, they are generally closed to the tools used for model generation. Test execution is performing in the same way for model-based testing as like traditional testing, so the same tools can generally be used. A list of model-based testing tools and the modeling notation they support is shown in below figure.

At present there are no standards that particularly support model-based testing; however there are different standards that can be used from other areas of software engineering. For instance, the most probably candidate for the test model is the new UML 2.0 standard from the (Object Management Group)

OMG [9]. Even though in its basic form this standard is most likely not precise enough for test modelling, it should be possible to improve this standard's usefulness with a test modelling profile. The output from the test generation tool is a test suite (or a sequence of test cases). The two most obvious candidates for the standard defining test cases are TTCN-3 [13] and a XML-based standard, such as that is used in the AGEDIS project [7]. If in the future model-based testing tools converse using standard data formats, then users will be able to select between combinations of tools for each of the three main stages rather than being tied to a single tool and supplier, with all the inherent economic and technological limitations that this brings.

The algorithms used for test creation and their equivalent coverage criteria are correlated by the same coverage measurement. There are some test coverage measures defined in BS 7925-2 [8], and even if these were originally defined to support component testing, some of the measures, such as state transition coverage, may be suitable for used in model-based testing. They are appears to be lack of in-depth knowledge in this area of the model-based testing approach. Some basic coverage measures are usually understood, but the more complex measures necessary for sensible coverage of large state models appear to be either proprietary, or in any case not well-understood by the testing community. For model based testing to become accepted on a wider scale, the test coverage achieved by the approach that should be both widely understood and accepted. Thus allowing model-based testing to be more likely compared with traditional testing and the best way for this to happen is for these measures to be consistent. At some point in the near future there will also come a requirement for a consistent model-based testing methodology, to provide a familiar understanding of the approach.

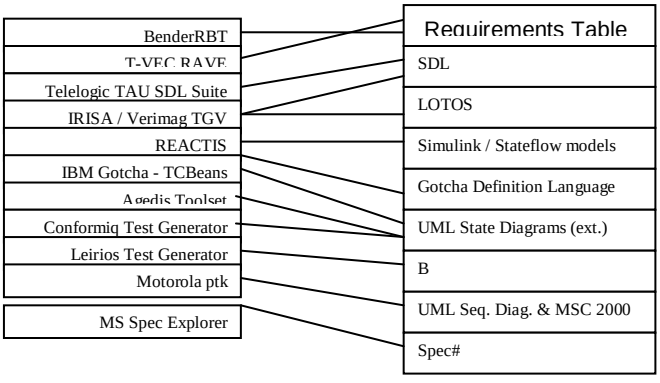

Figure 2: Model Based Testing Tools and Notations

IV. BENEFITS OF MODEL BASED TESTING

Many studies has been shown that model based testing is effective, mainly when used to test small applications, embedded systems, user interfaces and state-rich systems with practically complex data. Rosaria and Robinson interfaces, Agrawal (2000) studied testing graphical user and Whittaker (1993) embedded control software and Avritzer and Larson (1993) phone systems. Usually the mainly attractive attribute of model based testing is idea to be the automatic generation of test cases, but that is not all. Model of software may help refining uncertain and badly defined requirements [10]. By eliminating model defects before the coding starts and automating the test case creation the result is considerable cost savings and higher quality code. Other benefits that are more related to testing include e.g. the following, which were presented in [10]:

- **Comprehensive tests;** if the model is an entire abstraction of the software, it is likely to automatically generate test cases which cover each possible transition of the model by using graph algorithms.
- **Defect discovery**; model based test automation discovers defects more successfully than manual methods. The article established this with a case study in which manual method uncovered 33 defects in a system, and model based method all of those and in calculation 56 more. As it was shown in this section the benefits of model based testing are huge if modelling and all the related tasks are done powerfully, but it also has some difficulties and drawbacks.

DIFFICULTIES AND DRAW-BACKS OF MODEL BASED TESTING

Almost every research on model based testing agrees on one thing: deployment of model based testing into an organization requires considerable efforts and investments. In [12], the following three reasons for the desirable efforts and investments are offered:

- **Excessive amount of skills is necessary from the testers.** They need to be well-known with the model, which means knowledge of dissimilar forms of state machines, formal languages, and automata theory. In accumulation, expertise in tools and scripts is essential 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 chosen, different parts of software must be separated so that the modelling is easier because of the smaller areas and the actual model has to be built.
- **Models themselves have also several drawbacks.** The main one of those is the explosion of state-space required. Even a simple application can hold so many states that the maintenance of the model becomes complicated and tedious task.
- As can be shown from the list, model based approach to software testing is not the perfect solution. The positive side though is that all the points in the list can be overcome with thorough planning of the deployment of model based testing into an organization.

SUITABLE APPLICATIONS

Before anybody goes to using model-based testing, they must be sure that the approach is suitable for their circumstances.

The creation of test models is clearly a large up-front cost, but this must be recouped by the lower maintenance costs when the system is prepared. Certainly, if the system is estimated to have low maintenance costs, then the test modelling costs should not be recouped by the potential maintenance savings of model-based testing until much later in the maintenance phase. Low maintenance costs may be predicted, for example, if the system is planned to have only a short operational life or it is expected that there will be mainly few changes to the system required by the users (and its environment). Obviously the application must be appropriate for modelling in a supported notation. As stated in advance, switching applications have been found to be mainly suitable, as they are well-suited to modelling as a state model and there is good tool support for this [11]. The application should also be considered significant enough to warrant the cost of model-based testing. If high quality is not important to the customer then model based testing will be unlikely to be cost effective.

The complexity of performing traditional testing of simultaneous applications could also act as a driver towards using model-based testing, where the mix of many test cases and systematic test case generation provides good test coverage of complex models.

CONCLUSION

Model-based testing is a new and evolving technique that allows us to automatically generate software tests from explicit descriptions of an application's behaviour. Because the tests are generated from a model of the application, It is needed only update the model to generate new tests when the application changes. This makes model-based tests far easier to maintain, review and update than traditional automated tests. Testers who are willing and able to create model-based test programs can create flexible, useful tests for the cost of a general-purpose test language tool. Good software testers cannot avoid models. MBT has emerged as a useful and efficient testing method for realizing adequate test coverage of systems.Model-based testing has already been used effectively on a number of projects and the number of applications for which model-based testing will be appropriate will carry on to grow. A number of factors will make this growth. Second, more commercial quality tool support will become offered. Model-based testing will not, however, be suitable for all situations. There will still be projects where no explicit model of necessities is available and testing needs to be finished before the end of next week. There will also be situations where the available testers are not professional software engineers, but simply those re-assigned from other parts of the organization that are currently free.

REFRENCES

[1] Marius Nita, David Notkin "White-Box Approaches for Improved Testing and Analysis of Configurable Software Systems" IEEE 2009.

[2] Goutam Kumar Saha "Understanding Software Testing Concepts" ACM 2008.

[3] Practical Model-Based Testing: A Tools Approach, Mark Utting and Bruno Legeard, ISBN 978-0-12-372501-1, Morgan-Kaufmann 2007.

[4] Apfelbaum, Larry. "Model-Based Testing", Proceedings of Software Quality Week 1997.

[5] S. R. Dalal, A. Jain, N. Karunanithi, J. M. Leaton, C. M. Lott, G. C. Patton B. M. Horowitz, "Model-Based Testing in Practice", ICSE '99 1.0s Angeles CA Copyright ACM 1999 1-581 13-074-0/99/05.

[6] J. D. Musa, A. Iannino, and K. Okumoto. Software Reliability: Measurement, Prediction, Application. McGraw-Hill, 1987.

[7] Crichton et al, Using UML for Automatic Test Generation, 16th IEEE International Conference on Automated Software Engineering (ASE 2001), San Diego, USA, IEEE Computer Society, Nov 2001.

[8] BS 7925-2-1998, Software Component Testing.

[9] UML 2.0. See http://www.uml.org/.

[10] 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.

[11] Safford, E. Test Automation Framework, State-based and Signal Flow Examples. Twelfth Annual Software Technology Conference, Salt Lake City, USA, 2000.

[12] 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.

[13] TTCN-3.See http://www.etsi.org/ptcc/ptccttcn3.htm.