A Survey of Model-Driven Testing Techniques

Mohamed Mussa, Samir Ouchani, Waseem Al Sammane, Abdelwahab Hamou-Lhadj
Department of Electrical and Computer Engineering
Concordia University
1455 de Maisonneuve Blvd West
Montréal, QC, Canada
{mm_abdal, s_oucha, w_samman, abdelw}@ece.concordia.ca

Abstract

The model-driven approach to software development has not only changed the way software systems are built and maintained but also the way they are tested. For such systems, a model-based testing approach is much recommended since it is aligned with the new model-driven development paradigm that favors models over code with the objective being to reduce time to market while improving product quality. There has been a noticeable increase in the number of model-driven testing techniques in recent years. Although these techniques have a common objective they tend to vary significantly in their design. In this paper, we discuss the model-driven testing techniques presented in 15 different studies. We compare these techniques according to specific criteria including the modeling language used to represent the system design artifacts, the ability to automatically generate test cases, the testing target, and tool support.

Keywords: Software testing, model-driven software engineering, test case generation, software quality assurance

1. Introduction

The model-driven approach to software development is increasingly gaining the attention of both industry and academia. Unlike traditional development techniques which tend to focus on implementation, model-driven software development stresses the use of models at all levels of the software development process. The result of this shift has brought with it significant changes in the way software is designed, maintained, and tested.

Testing, which is the focus of this paper, has long been considered as a time consuming activity that calls for enhanced and powerful techniques. As stated by Baker et al. "Testing often accounts for more than 50% of the required effort during system development" [18]. A testing cycle encompasses three main parts: Test case generation, test execution, and test evaluation, with test case generation being perhaps the most complex and challenging part.

Model based development using the Unified Modeling Language (UML) [23] has led many researchers to using UML diagrams such as state machine diagrams, use-case diagrams, sequence diagrams, etc. to generate test cases. The major advantage of these model-based testing techniques is increased productivity and quality by shifting the testing activities to an earlier stage of the software development process and generating test cases that are independent of any particular implementation of the design [2].

There exist many model-based testing approaches and tools (e.g., [1, 2, 6, 9]), which vary significantly in their specific designs, testing target, tool support, and evaluation strategies. The objective of this paper is to survey and compare the techniques presented in more than 15 model-based testing approaches. We believe that the result of this survey can be used by practitioners as a reference work to understand the similarities and differences among these techniques, to determine the appropriate approach and corresponding tool that is most suited to their needs, or simply to reuse these techniques instead of reinventing them.

This paper is organized as follows: In Section 2, we describe our methodology for the survey. In Section 3, we present the surveyed testing techniques and discuss them according to the criteria set in Section 2. We conclude the paper in Section 5.

2. Methodology

There are several aspects of a model-driven testing technique that can be studied. In our study, we focus on the following criteria:

- **Modeling Language:** We discuss the modeling language (e.g., UML) targeted by the studied approaches.
- **Automatic Test Generation:** We discuss the test case generation mechanism used by the approach and whether it is automated or not. We also inspect the conditions that need to be satisfied for the test case generation mechanism to be effective.
- **Testing Target:** Although the surveyed studies focus on generating test cases from models, the target testing artifact varies from one approach to another - Some authors target the design models, whereas others target only the implementation.
- **Tool Support:** We provide information about the tools that support the studied approaches to generate the test cases. We also report on specific tool features where possible.

3. Model-Driven Testing Techniques

In this section, we first present the techniques surveyed in this paper. We then proceed to discussing these techniques, their similarities and differences, based on the aforementioned criteria. The section ends with a discussion on the presented studies as well as summary that features their main characteristics.

3.1. The Selected Approaches

The techniques discussed in this paper have been selected based on the numerous model-driven testing concepts they cover. Although the list of techniques does not represent all the studies that exist in the literature, we believe that it is representative of the state of the art in the field. The studies included in this paper are: Caverra et al. [1], Javed et al. [2], Baker and Jervis [3], Benjamin et al. [4], Born et al. [5], Bouquet et al. [6, 7], Farooq et al. [8], Crichton et al. [9], Ganov et al. [10], Hartmann et al. [11], Mingsong et al. [12], Schieferdecker [14], Schieferdecker et al. [15], Trong [16], and Yuan et al. [17].

3.2. Modeling Language

Yuan et al. present an automatic approach in [17] to generate test cases of a given business process of a web service. BPEL (Business Process Execution Language) [19] and UML activity diagrams are used to define the Process Under Test (PUT). The UML Testing Profile standard (UTP) [18] and the Testing and Test Control Notation Version 3 (TTCN-3) [14] concepts are used to construct the test case model. The UTP standard defines a framework for building concise test models that can be used to generate automatically test cases by applying the Model-Driven Architecture (MDA) approach [20] and possibly conformed transformation techniques. The generated test model can be tailored to target any of the following test types: unit testing, component testing, integration testing, or system testing. In addition to functional testing, UTP can target other types of testing domains like performance and efficiency testing.

The approach presented by Yuan et al. applies two automatic transformations to generate an executable test case set. The first transformation is used to build the Abstract Test Cases (ATC) from two models, the UTP model and the test case model. The second transformation is applied on the ATC to generate executable test case scripts, which are executed in the TTworkbench[1] environment. The authors' approach presents a practical application of the UTP framework.

Baker and Jervis present an approach that is similar to the previous one (i.e., relies on the UTP standard) [3]. In addition to generating test cases for validating the implementation, they provide a mechanism to validate the design model at early stages of the software development cycle. Timing and concurrency have also been handled by their approach. The approach has been successfully applied to many projects.

Unlike the aforementioned studies, Cavarra et al. in [1], and Crichton et al. in [9], present a test case generation approach that is not based on UTP. Their approach is based on extending UML using UML profiling capabilities. More precisely, they created two profiles. The first one is used to model the system under test (SUT) by extending class diagrams, object diagrams, and state diagrams to support testing properties. The other profile is used to capture the test directives (TD) which are composed of the object diagrams and state diagrams. Instances of these UML profiles can be built using any UML standard CASE tool capable to export the model as XMI. A transformation tool has been developed to map the SUT and TD to an intermediate format, which can be used to validate the design model by using the CAESAR/ALDEBARAN development package [24] to animate, verify, and model-check the intermediate format script. The script can also be used to produce a TTCN format which may be further translated to provide test cases.

Javed et al. present an interesting test development process [2], which utilizes the MDA initiative to generate automatically unit test cases. Their approach

[1] http://www.testingtech.com/products/ttworkbench.php

benefits from the existence of MDA's transformation tools to generate test cases. The authors define two types of transformations: Horizontal and vertical transformations. The horizontal transformation maps a Platform Independent Model (PIM) to another PIM using tools such as Tefkat [26] (a model transformation engine which is available as an Eclipse plug-in). The input model for this transformation is captured in a UML sequence diagram as sequence of method calls. The vertical transformation maps a PIM to a Platform Specific Model (PSM) using tools such as MOFScript (a model-to-text transformation language) [29]. It refines the produced model into a platform specific model of test cases. The target platform can be Java, JUnit, Smalltalk, SUnit. The authors also define a practical mapping process of their approach to the UTP standard.

Bouquet et al. present a model-based testing approach in [7], and a prototype tool in [6]. Their approach is based on a combination of a subset of UML (class, object, and state diagrams) and Object Constraint Language (OCL) [21] expressions to automatically generate test cases from these models. These diagrams and constraints are fed to a model-based test generator, LEIRIOS Test Designer [6], which generates test cases.

The authors discuss the need to alter the semantics of OCL to allow OCL expressions to have a side effect on the system state. They group the OCL expressions, especially the ones applied to post-conditions and guards, into active and passive expressions. They note that "it is necessary to introduce this active/passive operational interpretation of OCL into UML-MBT because of the lack of frame information in OCL" [7]. In their assumption, for example, the OCL expression *self.attr1 = self.attr2* will be treated as an active expression and the value of *self.attr2* will be assigned to *self.attr1*. One of the shortcomings of their approach is that it violates OCL semantics, which may hinder the acceptance of the approach by the UML community. One possible solution is to use an action language to express expressions that change the state of the system.

Farooq et al. provide a model-based regression testing approach [8]. Their approach assumes that the original design model of the system and the modified one (that is created after a change to the original model is made) are captured within a subset of UML diagrams, namely, class and state diagrams. The two model versions are compared semantically. Depending on the detected changes, the original test suite will be redefined and classified into three categories: Retestable, obsolete, and reusable. Retestable test cases have to be executed for regression testing as they relate to the system changes. Obsolete test cases are invalid for the updated version of the system, and they usually relate to elements of the system that have been deleted. Reusable test cases relate to unchanged parts of the system. They are valid but they are not required to be re-executed for regression testing.

Trong's approach, described in [16], offers a systematic procedure for testing UML designs. It uses class diagrams (with OCL expressions) and interaction diagrams. The initial test configuration is captured with object diagrams. The approach proceeds by executing the interaction diagrams and monitor the behavior of object diagrams to report any failure that may occur in comparison to the class diagrams and the OCL constraints.

Hartmann et al. present a technique that uses UML state diagrams and sequence diagrams to derive test cases for unit and integration level testing [11]. It also uses UML use cases and activity diagrams to derive test cases for the system level testing.

Mingsong et al. present an approach that uses UML activity diagrams to generate test cases for Java programs [12]. First, the approach generates random test cases, which are used to exercise the SUT. After that, the approach compares the running traces with the activity diagram to reduce the test case set. The author's approach, however, is limited to UML activity diagrams that do not contain concurrency or loops.

Born et al. introduce an interesting development method, named KobrA, based on MDA, UTP, and TTCN-3 [5]. The method provides a mechanism to develop the testing model in parallel to the functional model development. The authors extend the UML metamodel in order to accomplish their goal. The test model is generated in UTP, which can be executed by mapping it to existing test execution environments such as TTCN-3.

3.3. Automatic Test Generation

Automated software test generation greatly reduces the costs of software testing. Automatic test generation requires a specification language which has formal semantics such as Finite State Machines (FSM). In this section, we discuss the selected model-driven testing approaches with respect to their abilities to generate test cases automatically.

Javed et al. present a method based on sequence diagrams to generate unit test cases from a platform-independent model of the system [2]. First, they model the behavior of the system using sequence diagrams, which are then automatically transformed into a general unit test case model using model-to-model transformation rules. The resulting test case model is further transformed into concrete and executable test cases using additional transformation rules.

Bouquet et al. describe the LEIRIOS Smart Testing approach to the functional validation of a subpart of the StarUML case study [6]. Firstly, their approach models the SUT using UML/OCL, and then automates the process of design, generation, management and execution of a functional test suite. After, it publishes the generated test case. Finally, the executable scripts to automate the test execution on the SUT are generated.

Crichton et al. describe an architecture for model-based verification and testing using a UML profile in which projected (optimal) models are generated automatically for each specified purpose [9]. Class, object, and state diagrams are used to define essential and complete models from which test cases are generated. These projected models are translated automatically into a language of state machines, animated, verified, and used as a basis for automatic test generation.

Ganov et al. present a novel test generation approach based on symbolic execution to obtain data inputs and enumerate event sequences that can most likely lead to maximize code coverage of a GUI application [10]. There key contribution consists of the introduction of symbolic execution for GUI testing. They developed the Barad tool that can be used to perform automatic Java byte code instrumentation in order to generate the data for the data widgets of the GUIs.

Mingsong et al. use UML activity diagrams as design specifications and present an automatic test case generation approach [12]. The approach randomly generates abundant test cases from a Java program under testing. Then, by running the program with the generated test cases, they obtain the corresponding program execution traces. Finally, by comparing these traces with the given activity diagram according to the specific coverage criteria, they obtain a reduced test case set which meets the test adequacy criteria. The approach can be used to check the consistency between the program execution traces and the behavior of UML activity diagrams.

Trong defines a business process model which can be tested thoroughly and repeatedly whenever it is changed [16]. The proposed approach targets the generation of executable test cases from the given business process. The approach is composed of three stages: defining a process under test based on the business process model, generating abstract test cases from the process under test, and from abstract test cases to executable test cases. The test cases are transformed into test scripts in TTCN-3.

3.4. Testing Target

The testing target depicts the system artifact to which generated test cases are applied. Our study shows that model-based testing approaches can target early abstract UML design models, functional UML models, implementation units, or the complete system. More precisely, if the test cases are generated from early UML design models at the functional system design level then the test target is the functional UML design, for this case there are several ways to execute these tests. TTCN-3 is one of them. The test cases that are generated from the component specification level target the implementation of the component. JUnit is one of way to execute such test cases.

The authors of [2, 12] target the implementation. Mingsong et al. provide a way to generate test cases from activity diagrams and execute them on a java implementation [12]. Javed et al. generate test cases from sequence diagrams and propose a way to test the implementation using xUnit [2].

Several authors (e.g., [1, 3, 7, 9, 16]) target the UML model of the system in order to detect design faults. Some of these approaches even simulate the execution through the usage of an automatic generated intermediate format [1, 5, 9]. Some of the authors provide approaches to benefit from the TTCN-3 standards for the execution of test cases on the systems [6, 17]. Finally, the authors in [11, 14] provide approaches to test everything ranging from the single unit to the complete system's implementation.

3.5. Tool Support

From the state of the art, we have explored the different tools used and/or developed by the authors. Cavarra et al. compile the models written in the profiles into the Intermediate Format (IF) using the Intermediate Format language [1]. This new representation can be animated, verified, or model checked using the tools of the CAESAR/ ALDEBARAN Development Package (CADP) [24]. Also, it can be provided as input to the Test Generation with Verification (TGV) tool [25]. In the sequel of their work in [1], they adapt and combine TGV [25] and GOTCHA-TCBeans (Generator of Test Cases for Hardware Architectures) [28] in their AGEDIS test generation tool.

Javed et al. use an Eclipse Modeling Framework (EMF) based transformation engine to generate test cases from UML sequence diagrams [2]. They also model a sequence diagram as a sequence of method calls which is then automatically transformed into an xUnit model by applying model-to-model transformations using Tefkat [26]. In the second step, JUnit test cases are generated from the xUnit model by applying model-to-text transformations using MOFScript [29].

Bouquet et al. propose an original model-based testing approach, which is embedded in the LEIRIOS Test Designer tool [27] and is deployed in domains such as electronic transaction applications [7]. Using the LEIRIOS Smart Testing solution [27], they are able to provide HP Quick Test Professional adapter to manage and/or execute the generated test cases [6, 7].

Mingsong et al. implemented their approach into a tool prototype called AGTCG [12]. Its graphical interface allows the users to interactively construct, edit, and analyze activity diagram [12]. The tool can instrument Java programs according to the given activity diagrams, and use randomly generated test cases to run the instrumented Java program, and gather the corresponding program execution traces. By comparing the program execution traces with the activity diagram, the tool gives the test case sets which satisfy the special test adequacy criteria.

Baker et al. present a novel GUI testing framework called "Barad" based on symbolic execution [10]. Barad generates values for data widgets and enables a systematic approach that uniformly addresses the data-flow as well as event-flow for white-box testing of a GUI application. Barad complements the traditional approaches for GUI testing by providing a technique for testing a class of GUI applications that conventional approaches could not effectively verify.

In [4], Benjamen et al. use GOTCHA-TCBeans [28] as a prototype coverage-driven test generator to test a hardware model based on the MurØ verification system [22].

3.6. Summary

The majority of the surveyed approaches use the UML diagrams to build the test models. UTP (a UML profile for testing) has also been used by many researchers, and has been shown to be useful in model-based testing. We believe that UTP in combination with the MDA initiative not only can permit early testing of model-driven systems and ease the sharing of models between the system developers and the system testers.

Due to ambiguity of some parts of UML diagrams, most approaches use only a subset of UML. Perhaps, the OMG's initiative for providing executable UML (xUML) with formal semantics can strengthen the model-based testing since it will facilitate the testing of the whole system model without any restrictions.

While most of the presented approaches automatically generate and execute test cases, only one approach (Farooq et al. [8]) handles the evolution of the system design and requirement changes by focusing on regression testing by presenting a model-driven regression testing approach.

We have also noticed that out of the 15 techniques surveyed in this paper, only four of them use models as a testing target. All other techniques focus on testing the implementation although the test cases are generated from higher level models.

Some approaches such as Javed et al. [2] have fully benefited from the MDA paradigm for the automatic generation the test cases by using model transformation. In other words, concrete and executable test cases can be generated directed from the models by using model transformation rules that transform the design model into a model for expressing test cases. This eliminates the need for separate tool for the creation of test cases.

4. Conclusion

In this paper, we reviewed 15 testing approaches that focus on generated test cases from software models. We used various criteria for classifying these approaches such as the used modeling language, the automated aspect of the approach, the testing target, tool support, etc. We have built a comparison matrix to provide a clear view of the studied papers. The results of this paper can be used by software engineers to select a testing approach that best fit their needs. It can also be used by researchers in the field as a reference work that can help them build upon existing approaches.

5. References

[1] A. Caverra, J. Daves, J. Thierry, L. Mounier, A. Hartman, and S. Olvovsky, "Using UML for Automatic Test Generation", *In Proc. of the International Symposium on Software Testing and Analysis (ISSTA)*, 2002.

[2] A. Z. Javed, P. A. Strooper and G. N. Watson, "Automated generation of test cases using model-driven architecture", *In Proc. of the ICSE 2nd*

International Workshop on Automation of Software Test (AST), 2007.
[3] P. Baker and C. Jervis, "Early UML model testing using TTCN-3 and the UML testing profile", *In Testing: Academic and Industrial Conference Practice and Research Techniques, TAIC PART-Mutation 2007*, pp. 47-54, 2007.
[4] M. Benjamin, D. Geist, A. Hartman, Y. Wolfsthal, G. Mas and R. Smeets, "A study in coverage-driven test generation", *In Proc. of the 36th Conference on Design Automation Conference*, pp. 970-975, 1999.
[5] M. Born, I. Schieferdecker, H.-G. Gross, and P. Santos. "Model-Driven Development and Testing – A Case Study". *In Proc. of the 1st European Workshop on Model Driven Architecture with Emphasis on Industrial Application,* pp. 97-104, 2004
[6] F. Bouquet, C. Grandpierre, B. Legeard, and F. Peureux, "A Test Generation Solution to Automate Software Testing", *In Proc. of the 3rd international workshop on Automation of software test*, pp. 45-48, 2008.
[7] F. Bouquet, C. Grandpierre, B. Legeard, F. Peureux, N. Vacelet, and M. Utting, "A subset of precise UML for Model-based Testing", *In Proc. of the 3rd International Workshop Advances in Model Based Testing (AMOST)*, pp. 95-104, 2007.
[8] Q. Farooq, M. Z. Z. Iqbal, Z. I. Malik and A. Nadeem, "An approach for selective state machine based regression testing", *In Proc. of 3rd International Workshop Advances in Model Based Testing (AMOST),* pp. 44-52, 2007.
[9] C. Crichton, A. Cavarra, and J. Davies, "Using UML for Automatic Test Generation", *In Proc. of the Automation of Software Testing,* 2007.
[10] S. R. Ganov, C. Killmar, S. Khurshid, and D. E. Perry. "Test Generation for Graphical User Interfaces Based on Symbolic Execution". *In Proc. Proc. of the 3rd International Workshop on Automation of Software Test*, pp. 33-40, 2008.
[11] J. Hartmann, M. Vieira, and H. F. und Axel Ruder, "UML-based Test Generation and Execution", *White paper, Siemens Corporate Research*, 2004.
[12] C. Mingsong, Q. Xiaokang, and L. Xuandong, "Automatic Test Case Generation for UML Activity Diagrams", *In Proc. of the International Workshop on Automation of software test*, pp. 2-8, 2006.
[13] A. Pretschner. "Model-Based Testing", *In Proc. of the 27th international conference on Software engineering,* pp. 722 - 723, 2005.
[14] I. Schieferdecker, "A TTCN-3 based Test Automation Framework for HL7-based Applications and Components", *In Proc. of the 11th International Conference on Quality Engineering in Software Technology*, 2008.
[15] I. Schieferdecker, Z. R. Dai, J. Grabowski, and A. Rennoch, "The UML 2.0 testing profile and its relation to ttcn-3", *In Proc. of the 15th IFIP International Conference on Testing of Communicating Systems (TestCom),* pp. 79–94, 2003.
[16] T. T. D. Trong, "A Systematic Procedure for Testing UML Designs", *In Proc. of the International Symposium on Software Reliability Engineering in Denver*, 2003.
[17] Q. Yuan, J. Wu, C. Liu, and L. Zhang, "A model driven approach toward business process test case generation", *In Proc. of the 10th International Symposium on Web Site Evolution (WSE)*, pp. 41–44, 2008.
[18] P. Baker, Z. R. Dai, J. Grabowski, P. Haugen, I. Schieferdecker, C. Williams. *Model-Driven Testing: Using the UML Testing Profile*, Springer, 2007.
[19] M. Juric, P. Sarang, B. Mathew. *Business Process Execution Language for Web Services.* Packt Publishing, 2004.
[20] S. J. Mellor, K. Scott, A. Uhl, D. Weise. *MDA Distilled*. Addison-Wesley, 2004.
[21] J. Warmer, A. Kleppe. *The Object Constraint Language: Getting Your Models Ready for MDA*. Addison-Wesley, 2003.
[22] D.L. Dill, "The Murphy Verification System" *In Proc. of the Computer-Aided Verification Conference*, 1996.
[23] M. Fowler, K. Scott. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley. 1999.
[24] H. Garavel, F. Lang, R. Mateescu, and W. Serwe, "CADP 2006: A Toolbox for the Construction and Analysis of Distributed Processes", *In Proc. of the 19th International Conference on Computer Aided Verification,* pp. 158-163, 2007.
[25] J. R. Calamé, "Specification-based Test Generation with TGV", *Technical Report SEN-R0508, Centrum voor Wiskunde en Informatica*, 2005.
[26] M.J. Lawley and J. Steel, "Practical Declarative Model Transformation With Tefkat" *In Satellite Events at the MoDELS 2005 Conference, LNCS Vol. 3844*, 2005.
[27] E. Jaffuel, B. Legeard, "LEIRIOS Test Generator: Automated Test Generation from B Models", *Lecture Notes in Computer Science, Springer,* pp. 277-280, 2007.
[28] GOTCHA-TCBeans. IBM User Guide. URL: www.haifa.ibm.com/projects/verification/gtcb/documentation.html
[29] MOFScript User Guide: URL: http://www.eclipse.org/gmt/mofscript/doc/