ISSN 22773061

An Evaluation for Model Testability approaches

Pourya Nikfard[1], Suhaimi bin Ibrahim[2], Babak Darvish Rohani[3], Harihodin bin Selamat[4,] Mohd Naz'ri Mahrin5

Advanced Informatics School (AIS), University Technology Malaysia (UTM), International campus, Kuala Lumpur, Malaysia

{[1]npourya2@live.utm.my, [2]suhaimiibrahim@utm.my, [3]drbabak3@live.utm.my, [4]harihodin@ic.utm.my, [5]mdnazrim@utm.my}

Abstract- Design for testability is a very important issue in software engineering. It becomes crucial in the case of Model Based Testing where models are generally not tested before using as input of Model Based Testing. The quality of design models (e.g.; UML models), has received less attention, which are main artifacts of any software design. Testability tends to make the validation phase more efficient in exposing faults during testing, and consequently to increase quality of the end-product to meet required specifications. Testability modeling has been researched for many years. Unfortunately, the modeling of a design for testability is often performed after the design is complete. This limits the functional use of the testability model to determining what level of test coverage is available in the design. This information may be useful to help assess whether a product meets the target requirement to achieve a desired level of test coverage, but has little proactive effect on making the design more testable.

Council for Innovative Research

Peer Review Research Publishing System

Journal: INTERNATIONAL JOURNAL OF COMPUTERS & TECHNOLOGY

Vol 9, No 1

editor@cirworld.com
www.cirworld.com, member.cirworld.com

1. Introduction

Software testing is a significant part of both the software lifecycle and quality assurance activities. In simple form, testing is the software products dynamic execution in order to show the errors presence. Software testing is costly, and it is considered as the final barrier to the software product release. Model-based testing (MBT) is an automation of black box testing technique; its major difference from the conventional methods of black box testing is that the test cases are automatically produced by software tools that develop the expected behavioral models of the software under the test (SUT). MBT has

conveyed a lot of benefits. These contain errors early detection, which is measured to be very cheap. The major MBT liability is to knowing how to model SUT (Reza, Ogaard, & Malge, 2008).

Model-based testing (MBT) is a rising trend in test automation. In MBT, the SUT is modeled at an appropriate level of abstraction for testing, and tools are used to automatically create test cases based on this model. Given a set of appropriate methods and tools, MBT has been demonstrated to be a helpful and efficient means for high-level testing in diverse domains. Some benefits comprise decreased costs of maintenance in concentrating on a single high-level model, and boosted test coverage over the features expressed in the test model by the means of automated test generation (Puolitaival & Kanstrén, 2010).

Model-based testing (MBT) defendants the systematic use of software models in particular for testing activities, e.g. creation of test oracles, test execution environments and test cases. If a software model is well appropriate for activities of testing, we speak of a testable model. Thus, software models testability is a significant quality representing the degree to which a software model helps activities of testing in a given test context. The software models testability should already be analyzed and enhanced at modeling time before MBT activities start. This assists to save costs for the detection and correction of defects of testability in later stages.

For models used in model-based testing, their testability evaluation is a significant problem. Existing approaches lack some related features for a systematic and complete evaluation. Either they do (1) not think about the context of models of software, (2) not propose a systematic process for choosing and developing right dimensions,

(3) not describe a reliable and general understanding of quality, or (4) not separate between subjective and objective measurements (Voigt & Engels, 2008).

Testability is a property of program that is introduced with the purpose of forecasting efforts need for testing the program. To quantify the program testability is to relate the program with some numbers to present the degree of how easy are those definite testable properties in the program to be tested. High testability software means that errors could be discovered more easily during testing (if the software exist errors). Otherwise, it is complicated to be tested and is likely to be fewer reliable. Therefore the number of tests need in a program may be taken as the quantify of its testability. There has not been an instinctively acceptable measure of program testability. Software testability is the degree to which a software artifact (i.e. a software module, software system, design document or requirements) supports testing in a test context that given. Testability is not an inherent property of a software artifact and cannot be calculated directly (such as software size). Instead testability is an extrinsic property that results from software interdependency to be tested and the test goals, test methods used, and test resources (i.e., the test context). A minor degree of testability results in enhanced test effort. In tremendous cases a testability lack may hinder software testing parts or software requirements at all (Yeh & Lin, 1998).

2. Model Testability approaches

2.1 An Empirical Analysis of a Testability Model for Object-Oriented Programs

Authors presented a metric based testability model for object-oriented programs. The paper investigated empirically the relationship between the model and testability of classes at the code level. Testability has been investigated from the perspective of unit testing. They designed an empirical study using data collected from two open sources Java software systems for which JUnit test cases exist. To capture testability of classes, they used various metrics to quantify different characteristics of the corresponding JUnit test cases. In order to evaluate the capability of the model to predict testability of classes, they used statistical tests using correlation. The achieved results support the idea that there is a statistically significant relationship between the model and the used test case metrics (Kout et al., 2011).

2.2 Analysis of Object Oriented Complexity and Testability Using Object Oriented Design Metrics

The approach proposed in this paper revolves around the complexity and testability of object oriented design. Predicting complexity of design at class level helps to simplify the design as much as possible. Object oriented design metrics extended by the approach proposed in this paper helps to obtain the quantifiable results so that complexity of design can be predicted accurately. The approach is based on literature survey and the concepts developed are validated by mapping the metrics

model to the actual software projects. Literature survey concepts and estimated results obtained by actual projects verify each other. Improvements can be made to the proposed metrics. As more detailed designs are produced during later design phases, the proposed metrics can be applied to different other types of diagrams produced as a part of detailed design either in its existing form or by

modifying it according to requirements. Complexity analysis at coding stage can be performed by the object oriented design metrics proposed in this paper (Khalid et al., 2010).

2.3 Metric Based Testability Model for Object Oriented Design (MTMOOD)

The proposed model for the assessment of testability in object-oriented design has been validated using structural and functional information from object oriented software. The models' ability to estimate overall testability from design information has also been demonstrated using several functionally equivalent projects where the overall testability estimate computed by model had statistically significant correlation with the assessment of overall project characteristics determined by independent evaluators. The proposed model is more practical in nature having quantitative data on testability is of immediate use in the software development process. The software developer can use such data to plan and monitor testing activities. The tester can use testability information to determine on what module to focus during testing. And finally, the software developer can use testability metrics to review code, trying to find refactoring that would improve the testability of the code (Khan & Mustafa, 2009).

2.4 Automating regression test selection based on UML designs

The authors propose a methodology supported by a prototype tool to tackle the regression test selection problem at the architecture/ design level in the context of UML-based development. Their main motivation is to enable, in the context of UML-based development, regression test selection based on design change information, early in the change process. They

also present three case studies that were used as an initial feasibility and benefit assessment. These case studies are varied in the sense that they cover very different systems and changes, in both industrial and academic settings. Results show that design changes can have a complex impact on regression test selection and that, in many cases, automation is likely to help avoid human errors. Their objective has been to ensure that regression testing was safe while minimizing regression testing effort. But they have shown that certain changes may not be visible in the design and may require additional attention during coding or a special way to document them during design. Another limitation is that, based on UML design information, test selection may not be as precise as if it was based on detailed code analysis. Improving precision by analyzing in more detail guard conditions and OCL contracts is the focus of their current research. However, their case studies have only shown one case of imprecision in classifying a test case as retestable. Despite the above limitations, by providing a methodology and tool to perform impact analysis and regression test selection based on UML designs, the achievements are:

– Higher efficiency in test selection based on the automation of design change analysis and traceability between UML designs and regression test cases.

– Better support for assessing and planning regression test effort earlier in the change process that is once design changes have been determined (Briand et al., 2009).

2.5 Automatic generation of test specifications for coverage of system state transitions

The authors have presented a novel strategy for automatic state-based system testing for achieving transition path coverage. A severe handicap in system test generation for transition coverage arises due to the fact that system state models are not developed during any pragmatic development process. This may be attributed to the fact that the state models for practical problems tend to be extremely large and complex. To overcome this problem, they synthesize a system state model of an object-oriented system from the relevant UML models. The synthesized state model is then used to generate test specifications for transition coverage. Their approach to system testing does not replace traditional testing approaches such as, method coverage, message path coverage, etc. On the other hand, their approach to testing based on transition path coverage is intended to be used in a complementary manner with traditional test techniques. The test specification generated by our approach could detect bugs not detected using traditional testing, since the fault models for the two are different (Sarma & Mall, 2009).

2.6 Improving the Testability of Object Oriented Software through Software Contracts

The main goal of the study was to establish the relationship between software testability and software contracts. The software contracts reduce the testing effort and hence improve the testability of an object oriented class. The same is demonstrated through an example of a queue class written in C++. The results show that software contracts reduce the number of test cases

by 50% to test a class. Hence, the software practitioners can make use of software contracts to reduce the testing effort and hence improve the testability of the software (Singh & Saha, 2010).

2.7 Construction of a Systemic Quality Model for Evaluating a Software Product

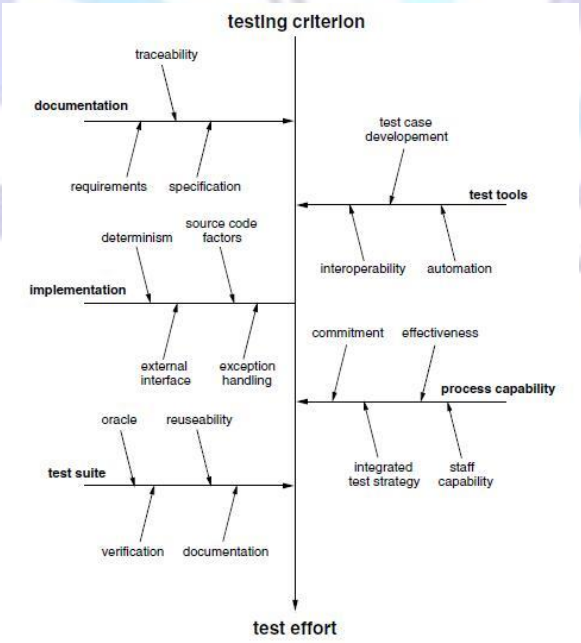
In referring to the well-known expression "build quality into software," Dromey points out that high-level quality attributes, such as reliability and maintainability, cannot be built into the software. What can be done though is to identify a set of properties (such as modules without side effects) and build them up consistently, harmoniously and fully to provide reliability and maintainability. Links must be forged between the tangible properties of the product and the high-level quality attributes. Dromey proposes three models, depending on the products resulting from each stage of the development process: requirements model, design model, and implementation quality model (programming). In comparing this model to ISO 9126, additional characteristics like process maturity and reusability are noticeable. It is important to point out the weighting Dromey gives to process maturity, an aspect not considered in previous models. This model seeks to increase understanding of the relationship between the attributes (characteristics) and the subattributes (subcharacteristics) of quality. It also attempts to pinpoint the properties of the software product that affect the attributes of quality. After analyzing various product quality models, the different quality attributes or characteristics found in each of them must be compared. Table 2.9 shows that the quality characteristics found in the majority of the models are: efficiency, reliability, maintainability, portability, usability and functionality, which have been present in more recent models. Because they are present in all the models studied, they can be considered essential and worthy of study (Ortega & Rojas, 2003).

2.8 An empirical study into class testability

Testability The ISO defines testability as''attributes of software that bear on the effort needed to validate the software product''. Binder offers an analysis of the various factors that contribute to a system's testability, which he visualizes using the fish bone diagram as shown in below Figure. The major factors determining test effort Binder distinguishes include the testing criterion that is required, the usefulness of the documentation, the quality of the implementation, the reusability and structure of the test suite, the suitability of the test tools used, and the process capabilities. Of these factors, our study is concerned with the structure of the implementation, and on source code factors in particular. We distinguish between two categories of source code factors: factors that influence the number of test cases required testing the system, and factors that influence the effort required to develop each individual test case. We will refer to the former category as test case generation factors and to the latter category as test case construction factors, which both are discussed below (Bruntink & Van Deursen, 2006).

2.9 Contract-Based Software Component Testing with UML Models

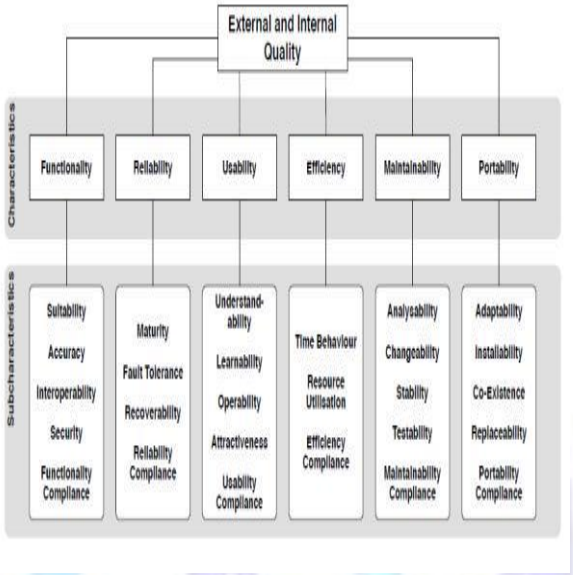
This research takes a different approach by incorporating testing-support artefacts (called test contracts) at the model-based specification level to improve model-based component testability with model-based test contracts. They believe that model-based testability stands at a testing level above traditional code-based testability, and thus is consistent with and supports model-based approaches to SCT (Zheng & Bundell, 2008).



ISSN 22773061

2.10 Quality Assessment and Quality Improvement for UML Models

The ISO/IEC 9126-model defines no uses, but distinguishes between internal quality, external quality and quality-in-use. The quality ISO/IEC 9126-model is a generic quality model that covers internal and external quality in one abstract model. The model for quality-in-use is similar to the operation use of the McCall model. However, quality-in-use and external quality are out of the scope of this paper, and therefore not discussed any further (Jalbani, 2011).



ISSN 22773061

3. Comparative evaluation

In this section we evaluate the above model testability approaches. We declare the achievement and main issue of each approach.

Row	*Authors*	Year	Method	Achievements	Main Issue
1	*Kout et al.*	2011	UML	1)investigate empirically the relationship between the model and testability of classes at the code level 2)Design an empirical study using JUnit 3)evaluate the capability of the model to predict testability of classes with using statistical tests 4)using correlation existing statistically significant relationship between the model and the used test case metrics	1)Accountability 2)Accessibility 3)Communicativeness 4)Not sufficient for Self-Descriptiveness 5)Not sufficient for both structural and behavioural architecture
2	*Khalid et*	2010	UML	1)Extend the object oriented design metrics 2)obtain the quantifiable results 3)predict complexity of design accurately	1)Accountability 2)Accessibility 3)Not sufficient for Self-Descriptiveness 4)Not sufficient for both structural and behavioural architecture
3	*Khan & Mustaja*	2009	UML	1)Validate model using structural and functional information 2)Demonstrate the models' ability to estimate overall testability from design information 3)The model is more practical in nature having quantitative data on testability 4)The software developer can use data to plan and monitor testing activities 5)The tester can use testability information to determine on what module to focus during testing 6)The software developer can use testability metrics to review code, trying to find	1)Accountability 2)Accessibility 3)Communicativeness 4)Not sufficient for Self-Descriptiveness 5)Availability of built-in test function 6)Test restartability 7)Not sufficient for both structural and behavioural architecture

				refactoring that would improve the testability of the code	
4	*Briand et al.*	2009	UML	1)tackle the regression test selection problem at the architecture/ design level in the context of UML-based development 2)Higher efficiency in test selection based on the automation of design change analysis and traceability between UML designs and regression test cases 3)Better support for assessing and planning regression test effort earlier in the change process that is once design changes have been determined	1)Accountability 2)Accessibility 3)Not sufficient for Self-Descriptiveness 4)Not sufficient for both structural and behavioural architecture
5	*Sarma & Mall*	2009	UML	1)synthesize a system state model of an object-oriented system from the relevant UML models 2)The synthesized state model is used to generate test specifications for transition coverage 3)The model is used in a complementary manner with traditional test techniques 4)The test specification generated by model could detect bugs	1)Accountability 2)Accessibility 3)Communicativeness 4)Not sufficient for Self-Descriptiveness 5)Not sufficient for both structural and behavioural architecture
6	*Singh & Saha*	2010	UML & Software contract	1)Software contracts reduce the testing effort 2)Software contracts improve the testability of an object oriented class 3)Software contracts reduce the number of test cases by 50% to test a class 4)Software practitioners can make use of software contracts to reduce the testing effort 5)Software practitioners can make use of software contracts to improve the testability of the software	1)Accountability 2)Accessibility 3)Communicativeness 4)Not sufficient for Self-Descriptiveness 5)Not sufficient for both structural and behavioural architecture
				1)requirements model, design model, and	1)Accountability 2)Accessibility

7	*Kojas & Urtega*	2003	Quality model	implementation quality model (programming) 2)notice process maturity and reusability 3)The model increase understanding of the relationship between the attributes (characteristics) and the subattributes (subcharacteristics) of quality 4)the quality characteristics are: efficiency, reliability, maintainability, portability, usability and functionality	3)Communicativeness 4)Not sufficient for Self-Descriptiveness 5)Retest efficiency 6)Not sufficient for both structural and behavioural architecture
8	*Deursen & Van Bruntink*	2006	Quality model	1)the usefulness of the documentation the quality of the implementation 2)the reusability and structure of the test suite 3)the suitability of the test tools used the process capabilities 4)factors that influence the number of test cases required testing the system 5)factors that influence the effort required to develop each individual test case 6)test case generation factors 7)test case construction factors	1)Accountability 2)Accessibility 3)Not sufficient for Self-Descriptiveness 4)Not sufficient for both structural and behavioural architecture
9	*Bundell & Zheng*	2008	Test contracts	1)Testability characteristics are: component traceability, component observability, component controllability, component understandability, and component test support capability 2)improve model-based component testability	1)Accountability 2)Accessibility 3)Communicativeness 4)Not sufficient for Self-Descriptiveness 5)Not sufficient for both structural and behavioural architecture
10	*Jalbani*	2011	UML	Distinguish between internal quality, external quality and quality-in-use	1)Accountability 2)Accessibility 3)Communicativeness 4)Not sufficient for Self-Descriptiveness 5)Not sufficient for both structural and behavioural architecture

4. Conclusion

This paper has aimed to provide an overview and compare recent progress in model testability. We study and survey several approaches. But we cannot claim that these approaches are comprehensive and exhaustive. Finally, we have comparison table with different columns that compares all the approaches. The main problem with most of these approaches to testing is considering the behavioral architecture in model testability. It should be noted that although the above comparison is conducted based on some prominent approaches, the outcome of this research is not restricted to such approaches. In other words, my considered criteria either can be served as features to be included in a newly developing system or may be applied to help generally evaluating or selecting model testability approaches. However the results of my comparison show that there is no single superior model testability approach in all cases. Therefore, deciding which approach to use in a certain scenario should be done based on its specifications and combine and present the new approach that is more comprehensive and mature is my future work.

5. Acknowledgement

This project is sponsored by Ministry of Higher Education (MOHE) in corporation with Universiti Teknologi Malaysia, Research Management Centre (UTM-RMC) under Vote No: 03H74. The authors also would like to thanks those who are directly or indirectly involved in this project.

6. Reference

Reza, H., Ogaard, K., & Malge, A. (2008). A Model Based Testing Technique to Test Web Applications Using Statecharts. *Fifth International Conference on Information Technology: New Generations (itng 2008)*, *0-7695-309*, 183–188. doi:10.1109/ITNG.2008.145

Puolitaival, O.-P., & Kanstrén, T. (2010). Towards flexible and efficient model-based testing, utilizing domain-specific modelling. *Proceedings of the 10th Workshop on Domain-Specific Modeling - DSM '10*, *978-1-4503*, 1. doi:10.1145/2060329.2060349

Voigt, H., & Engels, G. (2008). Quality Plans for Measuring Testability of Models. *Software Engineering* (pp. 1–16). Retrieved from www.vietnamesetestingboard.org

Yeh, P., & Lin, J. (1998). Software Testability Measurements Derived from Data Flow Analysis Tatung Institute of Technology. *2nd Euromicro Conference on Software*

Maintenance and Reengineering (CSMR'98) (pp. 1–7).

Kout, A., Toure, F., & Badri, M. (2011). An empirical analysis of a testability model for object-oriented programs. *ACM SIGSOFT Software Engineering Notes*, *36*(4), 1. doi:10.1145/1988997.1989020

Khalid, S., Zehra, S., & Arif, F. (2010). Analysis of object oriented complexity and testability using object oriented design metrics. *Proceedings of the 2010 National Software Engineering Conference on - NSEC '10*, 1–8. doi:10.1145/1890810.1890814

Khan, R. a., & Mustafa, K. (2009). Metric based testability model for object oriented design (MTMOOD). *ACM SIGSOFT Software Engineering Notes*, *34*(2), 1. doi:10.1145/1507195.1507204

Briand, L. C., Labiche, Y., & He, S. (2009). Automating regression test selection based on UML designs. *Information and Software Technology*, *51*(1), 16–30. doi:10.1016/j.infsof.2008.09.010

Sarma, M., & Mall, R. (2009). Automatic generation of test specifications for coverage of system state transitions. *Information and Software Technology*, *51*(2), 418–432. doi:10.1016/j.infsof.2008.05.002

Pourya Nikfard, Mohammad Hossein Abolghasem Zadeh, Suhaimi Bin Ibrahim (2013), A Comparative Evaluation of approaches for Web Application Testing, International Conference on Soft Computing and Software Engineering 2013 (SCSE'13), International Journal of Soft Computing and Software Engineering [JSCSE], ISSN: 2251-7545 & DOI: 10.7321/jscse, San Francisco, CA, USA.

Pourya Nikfard, Harihodin Selamat, Mohd Naz'ri Mahrin (2012), Functional Testing on Web Applications, Postgraduate Annual Research on Informatics Seminar (PARIS 2012).

Mohammad Reza Abbasy, Pourya Nikfard, Ali Ordi and Mohammad Reza Najaf Torkaman (2012), DNA Base Data Hiding Algorithm, International Journal on New Computer Architectures and Their Applications (IJNCAA) 2(1): 183-192 The Society of Digital Information and Wireless Communications, 2012 (ISSN: 2220-9085).

Mohammadreza Najaftorkaman ,Pourya Nikfard, Maslin Masrom, Mohammadreza abbasy (2011), An efficient cryptographic

protocol based on DNA chip, © 2011 Published by Elsevier Ltd. Selection and/or peer-review under responsibility of GCSE 2011, Procedia Engineering.

Mohammad Hossein Abolghasem Zadeh, Pourya Nikfard, Mohd Nazri Kama, Suhaimi Bin Ibrahim (2013), Software Changes: Related Software Artifacts and their Relationships, The Second International Conference on Advances in Computing and Networking (ACN - 2013), Bangkok, Thailand.

Mohammad Reza Najaf Torkaman, Pourya Nikfard, Nazanin Sadat Kazazi, Mohammad Reza Abbasy, and S. Farzaneh Tabatabaiee (2011), Improving Hybrid Cryptosystems with DNA Steganography, E. Ariwa and E. El-Qawasmeh (Eds.): DEIS 2011, CCIS 194, pp. 42– 52, 2011. © Springer-Verlag Berlin Heidelberg 2011.

Morteza Bagherpour, Pouria Nikfard, Arash Ghaed Mohammadi (2006), Hybrid Neural Network, Tabu search, Application to single machine scheduling with earliness and tardiness penalty, Proceeding of the V international conference "System Identification and Control Problems" SICPRO '06 Moscow'

Singh, Y., & Saha, A. (2010). Improving the testability of object oriented software through software contracts. *ACM SIGSOFT Software Engineering Notes*, *35*(1), 1. doi:10.1145/1668862.1668869

Ortega, M., & Rojas, T. (2003). Construction of a systemic quality model for evaluating a software product. *Software Quality Journal*, *11:3*(July), 219–242.

Bruntink, M., & Van Deursen, A. (2006). An empirical study into class testability. *Journal of Systems and Software*, *79*(9), 1219–1232. doi:10.1016/j.jss.2006.02.036

Zheng, W., & Bundell, G. (2008). Contract-Based Software Component Testing with UML Models. *Computer Science and its Applications, 2008. CSA '08. International Symposium on*, *978-0-7695*(13 - 15 October 2008), 83–102.

Jalbani, A. A. (2011). *Quality Assessment and Quality Improvement for UML Models*.