ISSN 22773061

A Comparative Evaluation of approaches for Model Testability

Pourya Nikfard[1], Suhaimi bin Ibrahim[2], BabakDarvish Rohani[3], Harihodin bin Selamat[4], MohdNaz'ri Mahrin[5]
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 importantissue 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 pro-active 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 Measuring design testability of a UML class diagram

In this paper, the authors identified two configurations in a UML class diagram that can lead to code difficult to test. These configurations are called testability antipatterns, and can be of two types, either class interaction or self-usage interaction. Those anti-patterns between classes may be implemented as interactions between objects in which case, the final software may be very difficult to test. The paper proposes a test criterion that forces to cover all object interactions. It also defines a model that can be derived from a class diagram, and from which it is possible to detect, in an unambiguous way all the anti-patterns. From this model, it is also possible to compute the complexity of anti-patterns which is the maximum number of object interactions that could exist (and should be tested). The testability measurement corresponds to the number and complexity of the antipatterns (Baudry&Traon, 2005).

2.2 Quality Plans for Measuring Testability of Models

In this paper, the authors focused on the evaluation of software models that are used for testing activities. They introduced the Model Quality Plan (MQP) approach for measuring quality of software models. They presented by an example how a MQP can be systematically developed for measuring the testability of UML statecharts. The MQP approach is based on a combination of the Goal Question Metric (GQM) and quality models. It consists of a top down process and a related metamodel. The process guides one how to develop a Model Quality Plan (MQP) and the metamodel states how a MQP may look like. The most important differences of their approach in respect to GQM are (1) adoption of the context characterization,

(2) integration of quality models, (3) refinement of measurement level, and (4) the formalization of their approach by an integrated UML model. Due to the documentation of a set of intermediate products and their high degree of details the initial effort for applying their approach remains heavy. Nevertheless, they are convinced that their approach is cost effective in the long term. Involving context factors tap the full potential to reuse existing quality plans. In addition, the systematic usage of context factors for the identification of information needs makes their approach more efficient. Last but not least the development effort can be considerably reduced by a dedicated tool support (Voigt & Engels, 2008).

2.3 Using UML Models and Formal Verification in Model-Based Testing

The authors presented an approach on combining UML modeling with formal verification in order to improve the quality of the models used for automated test derivation. While incorporating formal verification in the overall process gave number of advantages. Firstly, the quality of the models was improved allowing them to detect inconsistence in the models that were not detected by their custom OCL validation rules. Secondly, while modeling with UML-B, they observed several ambiguities, or not well-explained details, in the specifications that might have been difficult to observe by using UML only. In a way, by using formal specifications, they better understood the functionality of the SUT. However, it also increased the complexity of the model as we had to add more details just to prove the system correct. At the moment, traceability of requirements from test cases into formal models is not performed. However, this can be done by attaching textual requirements, taken from requirement model, to the generated Event-B specifications. This would certainly help to find which part of the system has passed or failed the

test and constitute a topic for our future work (Malik et al., 2010).

2.4 A Novel Quality Model for UML Models

In this paper, a continuous quality assessment and improvement methodology for UML have been proposed, which is based on quality model. The quality model uses inclusion relationship for three types of model Completeness, these are: incomplete, complete and executable UML models. The quality attributes of each quality attributes are adopted from a generic quality model ISO/IEC 9126. The three types of model completeness takes into account the different level of model completeness or different level of abstraction of UML model developed during software development phases. The purpose of the quality model is to provide a way to the modeler to select appropriate methods for continuous quality assessment and improvement of UML models (Jalbani et al., 2012).

2.5 Method for Improving Design Testability through Modeling

In this paper, authors mentioned that testability modeling is part of an overall strategy. In order to utilize testability modeling it is important to understand what role it will take in the products test strategy. With this understanding, a definition of what needs to be modeled and to what extent can be generated. From this, the testability modeling can be performed. The information yielded by the testability modeling can then be utilized as part of the design processes of investigation, analyzing alternatives, acting on the alternatives to yield the optimum balance of test within a product design. Documenting the testability decisions, actions and reasons behind them will provide information that is useful through the products lifecycle, potentially leading to reduced lifecycle costs (Emmert, 2010).

2.6 **Contract-Based Software Component Testing with UML Models**

The authors in this research extended the DbC concept to the SCT domain, and developed a new TbC technique with a key aim to bridge the gap between ordinary UML models (non-testable) and target test models (testable) and improve model-based component testability for effective UML-based SCT. Moreover, they introduced the new concept of test contract as the key testing-support mechanism, and the new concept of Contract for Testability as the principal goal of the TbC technique. They described the concept of test contracts based on basic component contracts, classified test contracts into internal and external test contracts for effective testing based on the new concept of effectual contract scope, and developed a set of TbC test criteria to realise testability improvement for achieving the CfT goals. Then, following the developed TbC working process, they put the TbC technique into practice to conduct UML-based CIT with the CPS case study and described and discussed contract-based component test model construction, component test design, and component test generation (Zheng&Bundell, 2008).

2.7 Research and Practice of the IVF Software Testability Model

Authors in this paper first analyze and review the research and practice of existing software testability models and various factors which affect software testability in software development process and then proposes an IVF (Iteration of Vector Factorization) software testability model based on years of statistics and analysis of test data of the Software Test and Evolution Center and validates it with practice. The IVF testability model is compact and practical and can directly be applied to the

practice of software engineering (Wri et al., 2010).

2.8 A measurement framework for object-oriented software testability

The framework presented in this paper provides practical and operational guidelines to help assess the testability of designs modelled with the UML. These guidelines are presented in such a way that the evaluation procedure can be tailored to the specific design and test strategies employed in a specific environment. From a research viewpoint, this paper presents a number of precise hypotheses that can be investigated through empirical means. In other words, it presents a starting point theory that can be verified and refined by experimental means (Mouchawrab et al., 2005).

2.9 Measuring Testability of Aspect Oriented Programs

The proposed model for the assessment of testability in aspect-oriented design has been validated. The models' ability to estimate overall testability from design information has also been demonstrated. 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 the design (Kumar, Sharma, &Sadawarti, 2010).

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	*Baudry & Traon*	2005	UML class diagram	1)Testability anti patterns 2) class interaction 3)self-usage interaction 4) cover all object interactions 5) defines a model that can be derived from a class diagram 6) the model is possible to detect, in an unambiguous way all the anti-patterns 7) the model computes the complexity of anti-patterns 8) The testability measurement corresponds to the number and complexity of the anti patterns	1) Accountability 2) Accessibility 3)Communicativeness 4) Self-Descriptiveness 5) Availability of built-in test function 6) Retest efficiency 7) Test restartability 8) not sufficient for both structural and behavioural architecture
2	*Voigt & Engels*	2008	Model Quality Plan (MQP)	1)measuring quality of software models 2) Goal Question Metric (GQM) 3) quality models 4) top down process adoption of the context 5) characterization integration of quality models 6) refinement of measurement level 7) the formalization of the approach by an integrated UML model 8) the approach is cost effective in the long term	1)Accountability 2)Accessibility 3)Communicativeness 4)Not sufficient for Self-Descriptiveness 5)Availability of built-in test function 6)Retest efficiency 7)Test restartability 8)Not sufficient for both structural and behavioural architecture
3	*Malik et al.*	2010	combining UML modelling with formal verificatio n	1)Improve the quality of the models 2)Automated test derivation 3)Detect inconsistence in the models that were not detected by custom OCL validation rules 4)Using formal specification with UML-B and have better 5)Understanding the functionality of the SUT	1)Accountability 2)Accessibility 3)Communicativeness 4)Not sufficient for Self-Descriptiveness 5)Availability of built-in test function 6)Retest efficiency 7)Test restartability 8)Not sufficient for both structural and behavioural architecture 9)Not sufficient for traceability of requirements

4	*Jalbani et al.*	2012	continuous quality assessment and improvement methodology for UML	1)The quality model uses inclusion relationship for three types of model Completeness, incomplete, complete and executable UML models 2)The quality attributes are adopted from a generic quality model ISO/IEC 9126 3)select appropriate methods for continuous quality assessment 4)select appropriate methods for improvement of UML models	1)Accountability 2)Accessibility 3)Communicativeness 4)Not sufficient for Self-Descriptiveness 5)Not sufficient for Availability of built-in test function 6)Retest efficiency 7)Test restartability 8)Not sufficient for both structural and behavioural architecture
5	*Emmert*	2010	products test strategy	1)design processes of investigation, analyzing 2)alternatives, acting on the alternatives 3)to yield the optimum balance of test within a product design 4)Documenting the testability 5)decisions, actions and reasons behind them 6)reduced lifecycle costs	1)Accountability 2)Accessibility 3)Not sufficient for Communicativeness 4)Not sufficient for Self-Descriptiveness 5)Not sufficient for Availability of built-in test function 6)Not sufficient for both structural and behavioural architecture
6	*Zheng&Bundell*	2008	TbC (Test by Contract) technique	1)bridge the gap between ordinary UML models (non-testable) and target test models (testable) 2)improve model-based component testability for effective UML-based SCT 3)introduce the new concept of test contract 4)introduce the new concept of Contract for Testability 5)describe the concept of test contracts based on basic component contracts 6)classify test contracts into internal and external test contracts for effective testing based on the new concept of effectual contract scope 7)develop a set of TbC test criteria to realise testability improvement for achieving the CfT goals	1)Accountability 2)Accessibility 3)Communicativeness 4)Not sufficient for Self-Descriptiveness 5)Not sufficient for both structural and behavioural architecture

7	*Wri et al.*	2010	Software development methods	1)propose an IVF (Iteration of Vector Factorization) 2)software testability model validate the model with practice 3)The IVF testability model is compact and practical 4)The IVF testability model can directly be applied to the practice of software engineering	1)Accountability 2)Accessibility 3)Not sufficient for Communicativeness 4)Not sufficient for Self-Descriptiveness 5)Not sufficient for both structural and behavioural architecture
8	*Mouchawrab et*	2005	UML	1)presents a number of precise hypotheses that can be investigated through empirical means 2)presents a starting point theory that can be verified and refined by experimental means	1)Accountability 2)Accessibility 3)Not sufficient for Self-Descriptiveness 4)Not sufficient for both structural and behavioural architecture
9	*Kumar, Sharma, & Sadawarti*	2010	UML	1)Validate a model for the assessment of testability in aspect-oriented design 2)Demonstrate the models' ability to estimate overall testability from design information 3)The proposed model is more practical in nature having quantitative data on testability 4)The software developer can use such data to plan and monitor testing activities 5)The tester can use testability 6)information to determine on what module to focus during testing 7)The software developer can use testability metrics to review the design	1)Accountability 2)Accessibility 3)Communicativeness not sufficient for Self-Descriptiveness 4)Availability of built-in test function 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 UniversitiTeknologi 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. *FifthInternational 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).

Baudry, B., &Traon, Y. Le. (2005). Measuring design testability of a UML class diagram.*Information and Software Technology*,

47(13), 859–879. doi:10.1016/j.infsof.2005.01.006

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

Malik, Q. a., Truscan, D., &Lilius, J. (2010).Using UML Models and Formal Verification in Model-Based Testing.*2010 17th IEEEInternational Conference and Workshops on Engineering of Computer Based Systems*, *978-1-4244*(22 - 26 March 2010), 50–56. doi:10.1109/ECBS.2010.13

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

Emmert, G. (2010). Method for improving design testability through modeling.*2010 IeeeAutotestcon*, *978-1-4244*, 1–4. doi:10.1109/AUTEST.2010.5613613

PouryaNikfard, Mohammad HosseinAbolghasemZadeh, 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.

DEIS 2011, CCIS

PouryaNikfard, HarihodinSelamat, MohdNaz'riMahrin (2012),

Functional Testing on Web Applications, Postgraduate Annual Research on Informatics Seminar (PARIS 2012).

Mohammad Reza Abbasy, PouryaNikfard, 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).

MohammadrezaNajaftorkaman ,PouryaNikfard, Maslin Masrom, Mohammadrezaabbasy (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 HosseinAbolghasemZadeh, PouryaNikfard, MohdNazri 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, PouryaNikfard, Nazanin Sadat Kazazi, Mohammad Reza Abbasy, and S. FarzanehTabatabaiee (2011), Improving Hybrid Cryptosystems with DNA

Steganography, E. Ariwa and E. El-Qawasmeh (Eds.):

194, pp. 42–52, 2011. © Springer-Verlag Berlin Heidelberg 2011.

MortezaBagherpour, PouriaNikfard, ArashGhaedMohammadi (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'

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

Wri, S., Congress, W., & Engineering, S. (2010).Research and Practice of the IVF Software Testability Model.*2010 Second WorldCongress on Software Engineering*, *978-1-4244*, 249–252.doi:10.1109/WCSE.2010.110

Mouchawrab, S., Briand, L. C., &Labiche, Y. (2005).A measurement framework for object-oriented software testability.

Information and Software Technology, *47*(15), 979–997.doi:10.1016/j.infsof.2005.09.003

Kumar, M., Sharma, P., &Sadawarti, H. (2010). Measuring testability of aspect oriented programs. *Computer Applicationsand …*, (Iccaie), 345–349. Retrieved fromhttp://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5735101