MANTECH Publications

Journal of Research in Computer Science and Engineering
Volume 8 Issue 3
ISSN No- 2457-0818

An Overview of Model-Based Testing Approaches Using Faceted Classification Framework

Maryam Al-Washahi [1], Yassine Jamoussi [2], Youcef Baghdadi [3]

Student [1], Associate Professor [2], Professor [3]

Department of Computer Science Engineering

Sultan Qaboos University

Corresponding Author's Email: *s135940@student.squ.edu.om [1], yessine@squ.edu.om [2], ybaghdadi@squ.edu.om [3]*

Abstract

Some studies have examined various model-based testing approaches and have presented an overview of these methods employing diverse techniques. Similarly, this paper reviews the state of the art of model-based testing using a different framework called faceted classification framework. This framework consists of four perspectives or views which are what, why, how, and which. Therefore, it has become easier to explore and compare the different approaches by using this framework as well as identify the gaps and weaknesses.

Keywords*-: Model-Based Testing, Classification Framework ,UML semantics ,Design Pattern.*

INTRODUCTION

Software testing is the process of validating and verifying the behavior of the software that is being tested. It plays a crucial part in the software development life cycle. It aims to create a highly uncompromising framework. Moreover, it emphasizes quality and identifies faults with that quality, which helps to gain the client's confidence. It is used also to determine whether or not the outcomes provided match the anticipated result and confirms that the system is error-free to produce more precise, trustworthy results. Software testing lowers the cost of development, makes maintenance easier, and ensures that the completed software properly fits the needs and demands of the user.

It is desirable to develop test cases at the design level to prevent costs and time wastage during software testing, improve the software's dependability, and enable the early detection of problems (Jena et al., 2014). Therefore, model-based testing is employed to find faults at the design stage. In model-based testing, the test cases are derived from a model that represents the functional requirements of the system being tested. Model-based testing is more effective and efficient since it combines code-based testing with the software's specification requirements (Alietal., 2007).

Automated testing based on UML models is one of the testing approaches and has become the de facto standard for software modeling (Bornstein, 2006). Moreover, UML automated-based testing is the intention of software testers. Thus, many researchers are attempting hard to integrate UML automated-based testing, to provide testers with methods that support and facilitate this type of testing. To comprehend the approaches taken by researchers, we must first understand the mechanism by which the UML automated-based testing was designed and implemented. Therefore, we require guidance that takes into account various aspects of the UML automated testing process.

LITERATURE REVIEW

Model based testing approaches have been reviewed by some studies which have been presented the overall of these approaches using different techniques. Utting (Utting et al., 2012) illustrated a taxonomy that covers six important dimensions of MBT. These dimensions are scope, characteristics, modelling paradigm, test selection criteria, technology and on/offline. “The definition of the process gives rise to six dimensions of MBT approaches. Along with possible instantiations of each dimension. The dimensions are largely independent of each other, but not entirely: for instance, if a project is concerned with a continuous rather than a discrete system, this is likely to limit its choice of modelling paradigm, of test selection criteria, and of test case generation technology” (Utting et al., 2012). In their study, Dias Neto (Dias Neto., et al, 2007) describes a systematic review performed on some model-based testing (MBT) approaches. These approaches compared against each other through some aspects which are representation models, support tools, test coverage criteria, the level of automation, intermediate models, and the complexity.

The main objective of Bernardino et al. (2017) systematic mapping study is to provide an overview on MBT tools and models used by those tools. They analyzed the MBT supporting

tools to identify their main features such as input models, model redundancy and the testing level.

RESEARCH APPROACH

The purpose of this paper is to propose a multidimensional framework for categorizing various approaches of MBT. This classification needs reviewing the domain content of MBT and classifying it into various perspectives/views. The faceted classification is the appropriate method for performing multidimensional classification since it can handle several approaches to categorization (Denton, 2003). The facet's versatility (Denton, 2003) enables it to be updated by adding new terms, changing current ones, or even deleting unneeded ones without affecting other facets (Al-Abri et al., 2016). This adaptability addresses the rapidly growing nature of MBT.

Faceted classification relies on describing attribute classes that can be constructed with various phrases. Regarding to Saipan et al. (2013), ‘‘these attributes have values that are defined within a domain, whereby a domain may be a predefined type such as integral or Boolean, an enumerated type ({x, y, z}), or a structured type (Set {x, y})” (Saidani et al., 2013).The proposed framework is similar to the one proposed by (Al-Abri et al., 2016; Al-Kalbani et al., 2015). The classification will be based on four distinct points of view, as seen in figure 1. Each point of view focuses on a distinct facet of the method. These four perspectives will be used to characterize each approach. What (topic), Why (purpose), How (method), and Which are the four points of view (tool). A view is made up of a number of facets that present a collection of attributes, which are determined by acceptable values by default.

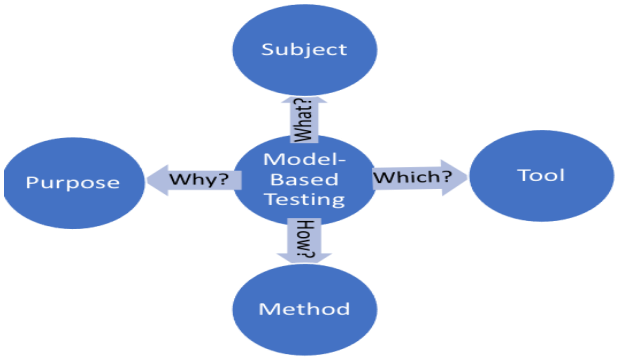

Fig. 1 The Classification Framework

Classifications Overview of the Framework

As previously stated, model-based testing approach will be classified based on four concepts or points of view: subject, purpose, method, and tool. Each view has a number of facets. Terms or qualities represent each facet.

Subject View

This view examines the frameworks "what" part. Describing what the approach is and what it is all about. Level of testing, input model, and SDLC phase are examples.

Input Model Facet

Input model is the first facet comes under subject view. At design phase software testers can make use of some of detailed design information as an input in MBT. There are various types of models which are used as an input in MBT. Basically, they can be categorized in two groups, non-UML notation models and UML notation models. First, non-UML Model notations includes (Thalheim, 2013):

- **Finite State Machines (FSM):** Finite State Machines model the behavior of a system or a component by representing it as a set of states, transitions, and events. They are often used for testing systems with discrete, event-driven behavior.
- **State charts:** State charts are an extension of FSMs and are used to represent complex systems with multiple concurrent states and transitions. They are particularly useful for modeling embedded systems and real-time applications.
- **Control Flow Graph (CFG):** The graphical depiction of control flow or computation throughout the execution of programs or applications.

Second, UML Model notations:

- **Class Diagram:** Represents the hierarchy or the structure of a system, including classes, their attributes, methods, and relationships. Moreover, it displays how classes are related to each other in terms of inheritance, associations, and dependencies.
- **Use Case Diagram:** It shows the interactions between a system and its users. Often used for capturing and documenting functional requirements.
- **Sequence Diagram:** It illustrates the dynamic behavior of a system by representing the interactions between objects with specific timeline.

- **State Machine Diagram:** It represents the behavior of an object or system as a finite state machine by showing system's states, transitions, and events that trigger state changes.
- **Activity Diagram:** It highlight the flow of activities within a system, showing the order of actions and decision points. Often used to model business processes, workflows, and use case scenarios.

Furthermore, along with these input models, additional techniques and knowledges can be used also at design phase. Design pattern is a one example of these knowledges. Design patterns provide practical reuse when software engineers encounter repeating issues. An effective use of design patterns results in the creation of software systems that are well-structured, maintainable, and reusable (Cepeda Porras et al., 2010). Finally, this facet could be defined as:

Input model: SET(ENUM{UML diagrams, non-UML notation, Design Pattern})

Level Facet

The second facet of subject view is level facet. It explains the approach's level of testing, for example functional testing, performance testing, security testing, etc. (Hooda et al., 2015). The set of this Facet is:

SET (Functional, System, Performance, Security)

SDLC Phase Facet

This facet specifies what is the SDLC phase where the software testing is performed. Testing can be performed at design level, development level or testing level. In summary, this facet can be declared as:

SET (Design, Development, Testing)

Purpose View

The next view in the classification framework of model-based testing is the purpose view. In this view the ‘‘why" side will be discussed of the approach. It shows in detail the objectives of model-based testing. There are various explanations and perspectives behind relying on

model-based testing. These purposes are also explored in terms of several facets which are describing in detail in the next sub-sections:

Testing Efficiency Facet

The usage of model-based testing is more effective solution for increasing the efficiency of software testing. Model-based testing usually is an automated process to substitute human behavior in testing a software and generating test result logs, thereby enhancing software product quality and lowering testing costs. Furthermore, model-based testing may reduce the number of generated test cases (Bums, 2014). So, this facet could be defined as:
SET {Time, Completeness, Cost}

Coverage Facet

Test coverage is referred to as a method for determining whether our test cases cover the program code and how much code is tested when those test cases are executed. There are various test coverage techniques. A test coverage criterion is essential for validating and analyzing test case test adequacy (Ali at el. 2007). They can also be used to control and terminate the test case generation process.

This section describes the eight most commonly used transition-based coverage criteria in test case generation (Utting et al., 2012):

- All-States Coverage requires a test case to visit every model state at least one time (Utting et al., 2012).
- All-Transitions Coverage requires that every transition be triggered at least once (Devroey et al., 2014).
- All-Transition-Pairs Coverage takes into account consecutive transitions that enter and exit a particular state sequentially. This coverage requires that each departing transition be triggered at least once for each state (Devroey et al., 2014).
- All-Configurations Coverage requires at least one visit to each configuration of the UML diagram (Utting & Legeard, 2010).
- All-One-Loop-Paths Coverage yields all paths that have at most one cycle; consequently, each generated path contains at most one repeating state (Muniz et al, 2015).

- All-Loop-Free-Paths Coverage must go over each loop path at least once. A loop-free path is one with no repetition
- All-Round-Trips Coverage is close to the all-one-loop-paths criterion in that it necessitates a test for each loop in the model.
- All-Paths Coverage when performing the abstract test case on it, it requires that each executable path be followed at least once (Devroey et al., 2014).

Method View

The method view explains how the framework works. It provides a solution to the question of how model-based testing is carried out. In other words, it explains the techniques used to demonstrate model-based testing.

Model Knowledge Facet

It demonstrates how the knowledge of the input model is formed. In short, the knowledge of the input model is formed either implicitly or explicitly and both semantics play a crucial role in defining the meaning and behavior of a model.

First, explicit semantics in a model refers to the clearly defined and documented meaning of the model elements. These semantics are explicitly specified in the model specification and the associated documentation, such as classes, associations, operations, states, etc. Moreover, they provide a precise description of how a model is constructed. They are essential for ensuring that the model is unambiguous and that it accurately represents the intended concepts and relationships within a system (France et al., 1998).

Second, implicit semantics in a model refer to the behaviors and constraints that are not explicitly specified in the model specification but are derived from the context and the way model elements are used. Which means these semantics may not be formally documented in the model specification but are still crucial for understanding and interpreting the model correctly (Cepeda Porras et al., 2010).

In the context of modeling, design patterns can be considered as implicit semantics that enhance the understanding and convey important architectural and structural information about a system. While UML diagrams provide a graphical representation of a system's

structure and behavior which are the explicit semantics, design patterns help convey recurring solutions to common architectural or design problems (Cepeda Porras et al., 2010). In summary, this facet can be described as:

Model knowledge: SET {Implicit semantics, Explicit semantics}

Test Generation Technology Facet

This facet explains the used technology to generate test cases. The following are examples (Utting et al., 2012):

- **Search-based algorithms:** the generated test cases are based on application of optimization and metaheuristic search techniques such as evolutionary algorithms (genetic programming), simulated annealing, etc.
- Theorem proving use of deductive theorem provers, supporting the generation of witness traces or counter examples, that check the satisfiability of a formula (guard on transitions, path condition, etc.)
- **Model checking:** model checking test generation technology involves using formal models of a system, typically expressed in a formal specification language, to automatically generate test cases or test scenarios.
- **Constraint solving:** represent the test case specification as a Constraint Satisfaction Problem (variables with finite domains, and associated constraints) that is instantiated to get a solution (or not).
- **Optimization algorithm:** these algorithms are employed to find the most efficient and effective test cases that maximize test coverage, identify defects, or achieve specific testing goals. Genetic algorithm, ant colony algorithm, simulated annealing algorithm are all examples.

Test generation technology: SET {search-Based, model checking, theorem proving, constraint solving, optimization-based}

Validation Facet

The approach validation method is explored in this facet. Here are two:

- **Case Study:** It is set of techniques, processes, and strategies employed to ensure the credibility, reliability, and trustworthiness of findings and conclusions derived from a case study research design.
- **Comparison:** Involves the process of validating research findings or results by comparing them to existing data, studies, or established standards.

Test case validation: SET {Case Study, Comparison}

Tools view

The tool view is the framework's ultimate view. A tool, or a collection of tools, is defined as the instrument required to achieve the task's goal based on the method used. The "which" of the comparison framework is described in this view. This view mentions the used tools in model-based testing in terms of two facets. These facets are model creation tools and test case generation tools.

Model Creation tools

This facet discusses the used tools to figure out the input of MBT which is the model. There are different tools for creating the model such as IBM Rational, Papyrus, EMF, etc (Utting et al., 2012).

Test Case Generation tools

This facet describes the test case generation tools, which are software applications designed to automatically create test cases based on various inputs, criteria, and requirements. These tools are widely used in software testing to improve test coverage, reduce manual effort, and increase the efficiency of the testing process. For example, Selenium, J unit, Model J Unit, Test NG, Acceleo, etc. (Rocha et. al., 2021).

COMPARISON OF FOUR MBT APPROACHES

Four MBT approaches have been selected for this study in order to use this framework as a comparison tool to study the different characteristics of these existing approaches. These approaches are selected based on the interest of review groups of different UML diagrams as input, know what the main intent of the testing is, explore diverse testing technology method and review different testing tools and libraries. Table (1) summarize the main contribution of

these four approaches. Table (2) provides a comprehensive summery of the faceted framework using these four approaches**.**

Table 1 Summary of the selected approaches

Ref	**Approach**	**Main Contribution**
Rocha et. al.(2021)	Model-based test case generation from UML sequence diagrams using extended finite state machine	The procedure starts with the construction of a sequence diagram and finishes with the generation of test cases for the scenarios described in the diagram. • Transformation between models: transforming Sequence diagram to EFSM • Stubs Generation: stubs source code is generated from the sequence diagram • Test Case Generation: Test cases are generated by EFSM-based test generation methods and ModelJUnit/JUnit libraries • Test Case Execution
Panthi et. al.(2018)	Functionality Testing of Object-Oriented Software Using UML State Machine Diagram	Use the state machine diagram to generate control flow graph of the program and accordingly produce all independent paths and test scenarios for the system. Step 1: Construct the state machine diagram using Rational Software Architect Step 2: Convert the state machine diagram into XML Step 3: create the .gv file which is readable by Graphviz using the XML file Step 4: Generate the control flow graph (CFG) of the system using the extracted information Step 5: Generate the test scenarios from CFG using Depth First Search Algorithm.

Basa et. al.(2018)	Genetic Algorithm-based Optimized Test Case Design Using UML	A test case generation and optimization technique from UML state chart diagram is proposed which is based on the total cost of each path traversed. • The proposed technique uses Genetic Algorithm (GA) for test sequence generation and optimization. • Then the test cases are generated from test sequences. • The generated test cases can be used to identify state-interaction and state-sequence faults using pre- and post- condition of events of transitions.
Verma et. al.(2014)	Automated Test case generation using UML diagrams based on behaviour	Proposed a new framework for automated test case generation from UML diagrams (Class Diagram, Sequence diagram and State chart Diagram and Use case Diagram) for object-oriented applications based on the behaviour: • Input Design Code Petal Files to Rational Rose • Read the Petal Files of Class, Sequence, State & Use Case Diagrams • If pattern found, Store it in Queue, Else Search for another pattern until end of File (EOF) • Create Text files from Queue and store it in database tables • Retrieve Strings to generate Test cases (Pattern Matching)

Table 2 Comprehensive summery of the faceted framework using the approaches

Views	Facets	Rochaet. al.(2021)	Panthi et. al.(2018)	Basa et. al.(2018)	Verma et. al.(2014)
1.Subject	• Input Model • Level of Testing • SDLC Phase	• Sequence Diagram • System testing • Design phase	• State machine • Functional testing • Design phase	• State Chart diagram • System testing • Design phase	• State Chart, Use case, Class, state chart diagrams • System testing • Design phase
2.Purpose	• Efficiency • Coverage	• Reduce number of test cases • All Paths & All states coverage	• To reduce the cost/time • All Paths & All states Coverage	• Help to uncover faults • All Paths coverage	• Speed up testing process • Transition pair coverage
3.Method	• Model Knowldge • Technology • Validate	• Explicit Semantics • model-checking (EFSM) • Comparision & case study	• Explicit Semantics • Search-Based algorithm (DFS) • Compariso n & case study	• Explicit Semantics • Search-Based algorithm (DFS) & Optimizatio n-based (Genetic) • Comparison & case study	• Explicit Semantics • model-checking • Case study
4. Tools	• Model Creation • Test Case	• EMF • Junit, ModelJunit,	• Rational Software Architect	• Rational Software Architect	• Rational Rose • -

	Generation	Acceleo, Selenium	• Graphiz, SAX Parser	• -	

DISCUSSION

Software testing is a crucial step in the software development process, and its main objective is to find bugs or other problems in the program to make sure it works as intended and fulfills user needs and expectations. Researchers and software testing practitioners select models for software testing because the use of models in software testing is driven by the need for more efficient, systematic, and comprehensive testing processes, especially in the context of complex software systems. As shown in Table 2, the four approaches give general overview about the state of art in MBT. They can be summarized as follow:

Subject View

- As Table 2, illustrates, most of the approaches considered UML diagrams as input model. Most of the selected approaches specified a single input model, while (Verma et. al., 2014) specified a group of input models.
- Functionality testing or system testing were the level of testing for these approaches and all of the approaches conducted testing process at design phase.

Purpose View

- In term of efficiency purposes, the mentioned approaches focused on reducing number of test cases, time and cost and to speed up testing process.
- While their coverage purposes were to enhance all path coverage, all test coverage and transition pair coverage.

Method View

- All of the selected approaches formed the knowledge of the input model by specifying its explicit semantics regardless the implicit semantics.
- The test case generation technologies were search based algorithm which is Depth First Search, model-checking and optimization-based algorithm. Moreover, their approaches have been validated by case studies and comparisons.

Tool View

- Eclipse Modling Framework, Rational Software Architect and Rational Rose were the selected tools to create the input models.
- Moreover, there are other tools selected by the four approaches which are helpful in generating test cases such as, Junit, ModelJunit, Acceleo, Selenium and Graphiz.

CONCLUSION

A comprehensive overview of the state-of-the-art in Model Based Testing has been demonstrated by adopting the MBT comparison framework which is based on four different views (subject, purpose, method, and tool). From the study, we can conclude that, the field of MBT is a wide in term of the selected model, the purpose of testing, the used method and the supported tools. Despite the diversity in those views, each approach aims at a single, specific target which is automating the process of generating test cases from models.

However, the existing model-based testing approaches don't take into consideration the specificity the implicit semantics. Moreover, some of the generated test cases is not efficient in terms of completeness and there is some overlapping. Also, some of the generated test cases need to enhance some coverage criteria. Therefore, an approach using the four views suggested in this study is required to satisfy all of these requirements.

ACKNOWLEDGEMENTS

The authors express their gratitude to those who provided assistance throughout the course of this research, with special recognition for the academic and technical support received from the Department of Computer Science within the College of Science at Sultan Qaboos University.

REFERENCES

1. Al-Abri, A., Jamoussi, Y., Kraiem, N., & Al-Khanjari, Z. (2017). Comprehensive classification of collaboration approaches in E-learning. Telematics and Informatics, 34(6), 878-893.
2. Al-Kalbani, J., Kraiem, N., Al-Khanjari, Z., Jamoussi, Y., 2015. Towards a comprehensive view of web services QoS models. Int. J. Multimedia Ubiquitous Eng.10 (10), 259–272.

3. Ali, S., Briand, L. C., Rehman, M. J. U., Asghar, H., Iqbal, M. Z. Z., & Nadeem, A. (2007). A state-based approach to integration testing based on UML models. Information and Software Technology, 49(11-12), 1087-1106.
4. Basa, S. S., Swain, S. K., & Mohapatra, D. P. (2018). Genetic algorithm based optimized test case design using UML. Journal of Computer and Mathematical Sciences, 9(9), 1223-1238.
5. Bernardino, M., Rodrigues, E. M., Zorzo, A. F., & Marchezan, L. (2017). Systematic mapping study on MBT: tools and models. IET Software, 11(4), 141-155.
6. Bums. D, Selenium 2 Testing Tools: Beginner's Guide, Packt Publishing, 2014, pp.12-13.
7. Burnstein, I. (2006). Practical software testing: a process-oriented approach. Springer Science & Business Media.
8. Cepeda Porras, G., & Guéhéneuc, Y. G. (2010). An empirical study on the efficiency of different design pattern representations in UML class diagrams. Empirical Software Engineering, 15(5), 493-522.
9. Denton, W., 2003. How to Make a Faceted Classification and Put it on the Web. See <http://www.miskatonic.org/library/facet-web-howto.html>.
10. Devroey, X., Perrouin, G., Legay, A., Cordy, M., Schobbens, P. Y., & Heymans, P. (2014). Coverage criteria for behavioural testing of software product lines. In Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change: 6th International Symposium, ISoLA 2014, Imperial, Corfu, Greece, October 8-11, 2014, Proceedings, Part I 6 (pp. 336-350). Springer Berlin Heidelberg.
11. Dias Neto, A. C., & Travassos, G. H. (2008, October). Surveying model based testing approaches characterization attributes. In Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement (pp. 324-326).
12. France, R., Evans, A., Lano, K., & Rumpe, B. (1998). The UML as a formal modeling notation. Computer Standards & Interfaces, 19(7), 325-334.
13. Gaffar, A., & Moha, N. (2005). Semantics of a pattern system. STEP, 2005, 211.
14. Jena, A. K., Swain, S. K., & Mohapatra, D. P. (2014, February). A novel approach for test case generation from UML activity diagram. In 2014 International Conference on

Issues and Challenges in Intelligent Computing Techniques (ICICT) (pp. 621-629). IEEE.

15. Li, H., & Lam, C. P. (2005, September). An ant colony optimization approach to test sequence generation for state based software testing. In Fifth International Conference on Quality Software (QSIC'05) (pp. 255-262). IEEE.
16. Muniz, L., Netto, U. S., & Maia, P. H. M. (2015). A Model-based Testing Tool for Functional and Statistical Testing. In Proceedings of the 17th International Conference on Enterprise Information Systems (ICEIS) (pp. 404-411).
17. Panthi, V., Gardizy, A., Mohapatra, R. K., & Mohapatra, D. P. (2018, December). Functionality testing of object-oriented software using UML state machine diagram. In 2018 International Conference on Circuits and Systems in Digital Enterprise Technology (ICCSDET) (pp. 1-7). IEEE.
18. Rocha, M., Simão, A., & Sousa, T. (2021). Model-based test case generation from UML sequence diagrams using extended finite state machines. Software Quality Journal, 29, 597-627.
19. Saidani, O., Kaabi, R.S., Kraiem, N., Baghdadi, Y., 2013. A multidimensional framework to classify goal-oriented approaches for services. In: Computer Applications Technology (ICCAT), 2013 International Conference on. IEEE, pp. 1–5.
20. Thalheim, B. (2013). Entity-relationship modeling: foundations of database technology. Springer Science & Business Media.
21. Utting, M., & Legeard, B. (2010). Practical model-based testing: a tools approach. Elsevier.
22. Utting, M., Pretschner, A., & Legeard, B. (2012). A taxonomy of model-based testing approaches. Software testing, verification and reliability, 22(5), 297-312.
23. Verma, A., & Dutta, M. (2014). Automated Test case generation using UML diagrams based on behavior. International Journal of Innovations in Engineering and Technology (IJIET), 4(1), 31-39.