International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9
***Article Received:** 25 July 2023 **Revised:** 12 September 2023 **Accepted:** 30 September 2023*

Model-Based Testing Approaches using UML Diagrams: A Systematic Literature Review

[1]Jyoti Gautam Tiwari, [2] Ugrasen Suman
[1]Department of Computer Science and Engineering SGSITS, Indore, M.P., India
jyotitiwariscs@gmail.com
[2]Professor, School of Computer Science& IT Devi Ahilya University, Indore, M.P., India
ugrasen123@yahoo.com

ABSTRACT: Software Unit Testing (SUT) is the starting point for Model-Based Testing (MBT), a testing method. The Unified Modeling Language (UML) has become the standard for modelling software in professional and academic settings. There are various uses for the modelling language known as UML. The findings of an SLR on UML-based model-based testing methodologies are presented in this paper. Thirty-five primary articles about six research issues were examined using selection and exclusion criteria. Methods, model class, intermediate format use, and testing methodology are the primary points of examination. The review outcomes identify future research needs and avenues of inquiry.

Keywords – *Model-Based Testing, Unified Modeling Language, Systematic Literature Review.*

INTRODUCTION

Software engineering applies engineering principles to software creation, deployment, upkeep, and eventual retirement [1]. Software testing aims to identify issues with a program by running it. It is an essential part of the software industry and a necessary step in software development. Test cases are created, run, and evaluated as part of software testing's primary focus [2]. Among the many benefits of testing analysis and design models is the opportunity to discover test cases earlier.

When necessities are being established, it is via early testing that analysts and designers get a deeper comprehension of requirements, finds better ways to communicate those needs, and verify that the conditions they have defined are, in fact, testable—saving time, money, and energy by finding defects early on in the development process. Before a project, the test cases are checked to ensure accuracy. There is always a problem with the precision of test cases, particularly system test cases [3]. Programmers, testers, and managers agree that the analysis and design phase is crucial for choosing testing approaches based on work principles, the link between design and requirements, the frequency of specification change, and conditions [4].

While object-oriented testing techniques are similar to traditional testing, challenges arise due to specific aspects of object-oriented programming that arise when actually putting these techniques to the test [5, 6]. Software testers often use one of three primary approaches: code-based testing, specification-based testing, or model-based testing.

Software testing's objective is to confirm that an application's features and functionalities meet the standards outlined in the software requirement specification [7]. More than half of the entire time spent on software development is devoted to testing, one of the most critical aspects of the Software Development Life Cycle SDLC [8]. When software systems grow in size and complexity, test case design becomes one of the trickiest parts of the testing [9]. Automatically generating test cases may improve reliability and performance while decreasing development costs [10].

In Model-Based Testing (MBT), test cases are produced in whole or in part based on a model that defines certain (often functional) features of the SUT. Tools for model-based development include techniques like code generation and simulation [11]. The steps involved in UML MBT are shown in Fig. 1. From the Software Requirement Specification (SRS), a model of the SUT is built.

The same holds true for writing the test report, which is based on the criteria used to choose test cases that show flaws, mistakes, and likely failures. If test selection criteria are applied to the specification, the system model is then utilised to produce the actual test cases. The SUT is used to run the tests, and the results are analysed. Here rectangle box represent the artifacts whereas rounded corner rectangle box represent the process.

The combination technique used by MBT to test the program uses both the specification and the source code, making it more effective and operational than code-based methodologies. MBT's primary areas of study are in ensuring that programs are consistent with their underlying architectures and in validating

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9
***Article Received:** 25 July 2023 **Revised:** 12 September 2023 **Accepted:** 30 September 2023*

the accuracy of the models used to create such programmes. Model-based testing tests the software using the model. The typical MBT procedure begins with a model description, continues with the definition of test requirements, and then moves on to developing test cases grounded in the model, followed by their execution, review, and eventual conclusion. An imperfect kind of MBT, model-based mutation testing guarantees that the generated test cases will discover a subset of incorrect specification implementations. Since mutations can be used as stand-ins for wrong requirements, the acronym MBMT describes this process well.

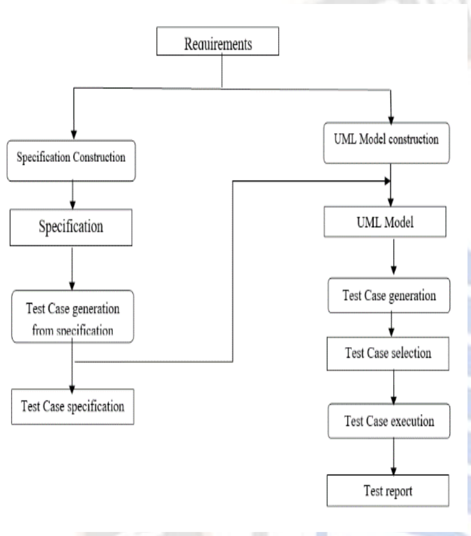

Figure 1. Process of UML MBT

Models are used in Model-Based Testing (MBT) for the purpose of automatically creating and running test cases [12].A model is only a symbolic depiction of the ideal state that an SUT should achieve. Several system representations may be utilized with MBT to produce test cases for different facets of the SUT. Existing models include the UML, Finite State Machines (FSMs), Petri Nets, I/O automata, and Markov Chains[17]. In recent years, MBT has seen an increase in the usage of software models for the creation of test cases, either manually or automatically [13].

The Object Management Group established UML 1997. UML is a modeling language fused to create graphical representations of software systems (UML). These models allow for assessing and analysing potential designs, which can aid in achieving design goals. From a UML model, test cases can be constructed using a variety of methods. Recently, the testing community has become increasingly interested in the use of UML diagrams for the modeling of object-oriented software systems [14,15]. Despite their lack of clarity, UML diagrams include crucial details for making test cases and give direction for the automated development of test cases. This makes the process of developing test cases in UML an integral and difficult aspect of MBT [16].

The research on MBT is dispersed, with diverse studies covering both the common and the variant techniques and the recommendations and lessons gained.

The following questions are discussed in this paper:

1. What are the different methods used in UML MBT?
2. What UML models are used in different methods in the primary study?
3. Is the intermediate form used in the primary study?
4. Which testing level is achieved in the primary study?
5. What are the coverage criteria used in the primary study?
6. Is the primary research include a case study?

In order to provide answers to the aforementioned questions, SLR is conducted. Using a systematic review process, we combed through 65 publications and determined that 35 represented high-quality primary research that addressed our concerns. On the basis of this investigation, we present the identified difficulties and explain the main trends and methods in data extraction.
The format of this essay will be as follows. In the second section, we present some context and relevant studies. Methods for this inquiry are laid out in Part 3. In Part 4, we show the SLR findings, including the problems and their remedies as seen in the UML MBT; in Section 5, we debate these findings; and in Section 6, we offer some last thoughts.

BACKGROUND AND RELATED WORK

At more abstract level of a system, graphs are used to represent UML diagrams. The data must meet certain route constraints during the way from the beginning to the ending node. You may generate test cases from graphs by following these steps: defining the requirements, constructing the graph, determining the test requirements, and selecting pathways to cover the requirements.

UML has a comprehensive library of diagrams and notations that may be used in a wide variety of contexts. The meta-model allows for the semantics of the diagrams to be interpreted in a variety of ways by various UML tools. In order for MBT tools to understand UML models, a subset of UML must be selected, and its semantics clarified. Each MBT solution takes a somewhat different tack by allowing for a unique collection of diagram types and establishing a subset of those diagrams that is guaranteed to be safe for usage in models. It's important to specify not just the static data portion of the model, but also its dynamic behavior.

To create test cases from UML diagrams, Kunxiang Jin et al. used SLR [19]. Test case generation using UML diagrams has been found to have some difficulties, and solutions to these problems have been presented. One of the important issues in UML diagrams is state explosion due to loop transition or events. By identifying loop transitions and events, model slicing technique can be applied to divide the original diagram into main diagram and sub-diagram. They provide a strategy for optimizing test cases by prioritizing test cases. Several methods failed to provide executable test cases because of the unknown nature of the system as actually implemented. Methods for transforming models may be utilized to address the implementation issue.

Tanwir Ahmad et al. used UML activity diagrams to perform SLR on MBT [23]. Researchers have called attention to the fact that the majority of suggested methodologies need to be thoroughly assessed before conclusions based only on the study are made. The majority of the methods offered have been designed for usage in a specific subset of the application domain, and the non-functional needs of the SUT have yet to be tested using UML ADs. Most studies only cover a small portion of the MBT procedure, seldom explaining how the offered methods fit into the overall development process.

RESEARCH METHOD

"A systematic literature review, also known as a systematic review, is a type of secondary study that utilises a well-defined methodology to identify, analyse, and compile all existing studies related to a specific research question in a manner that is objective and, to some extent, repeatable," as defined by the authors of one such review. While several MBT methods have been presented, no one has yet made a concerted attempt to compile a comprehensive review of the relevant literature. Consequently, the literature on MBT's efficacy, common methodologies, methods, and learnt recommendations and findings, are dispersed over several studies.

Research Questions

The following study questions were developed to help provide light on the difficulties and possibilities inherent with UML MBT.
RQ1: What are the Different Methods Used in UML MBT?
RQ2: What are the UML models are used in different methods in the primary study?
RQ3: Is intermediate form is used in the primary study?
RQ4: Which testing level is achieved in the primary study?
RQ5: How was the primary research conducted?
RQ6: Is the primary study including a case study?

Data Sources

Many references are provided below. The following has been selected for use in UML MBT.:

- IEEE Xplore
- ACM Digital library
- Wiley InterScience
- Elsevier Science Direct
- Springer

Data Retrieval

We have used the following Search string for UML MBTs :
(("Test case generation" OR "UML behavioural model test case generation" OR "UML behavioural model testing," OR "static model testing," OR "UML testing," OR "test coverage analysis," OR "MBT") AND ("model-based testing," OR "optimise test case generation using UML model" OR "Tsting UMl model")).

Studies Selection

The following requirements must be fulfilled for studies to be deemed primary: they must be written in English, published online between 2005 and 2020, mention UML MBT, and expressly choose one of the UML models for test case creation or optimization.

Table 1: Number of primary studies from different sources

Resources	**Search results**
IEEE Xplore	**9**
ACM Digital Library	**8**
Elsevier Science Direct	**7**
Springer Link	**6**
Wiley InterScience	**3**
Web of Science	**2**
Total	**35**

The search criteria have been adjusted to better suit the needs of the investigation. Table 1 displays the outcome of applying the search string and the results of the search.

Data Extraction

After reading 65 papers as primary study. Out of a total of 65 publications, 30 were thrown out because they either repeated previous research or they didn't use any of the test case creation or optimisation tools that are part of the UML MBT.

Title, authors, publication year, case study, optimisation approach, input model, research methodology, and results are

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9
Article Received: *25 July 2023* ***Revised:*** *12 September 2023* ***Accepted:*** *30 September 2023*

used to compile the data. Information was collected for 35 articles. The data extraction contains the no. of study, publication year, Input model, Intermediate form, testing type and case studies.

Threats to Validity

Despite our efforts, we can only explore some of the potential data sources for UML MBT research in this paper. We have included most of the phrases used to express UML MBT and model-based Architecture since different databases use various jargon. To limit the amount of unrelated research, the search term is tweaked significantly to account for search engine preferences. We have performed the search procedure for identifying applicable studies in UML MBT in a methodical and comprehensive manner. Time constraints and the ever-increasing volume of research in this area mean that some publications may have needed to be included.

RESULTS

Year-wise distribution of studies

Fig. 2 represents the year-wise distribution of the research paper.

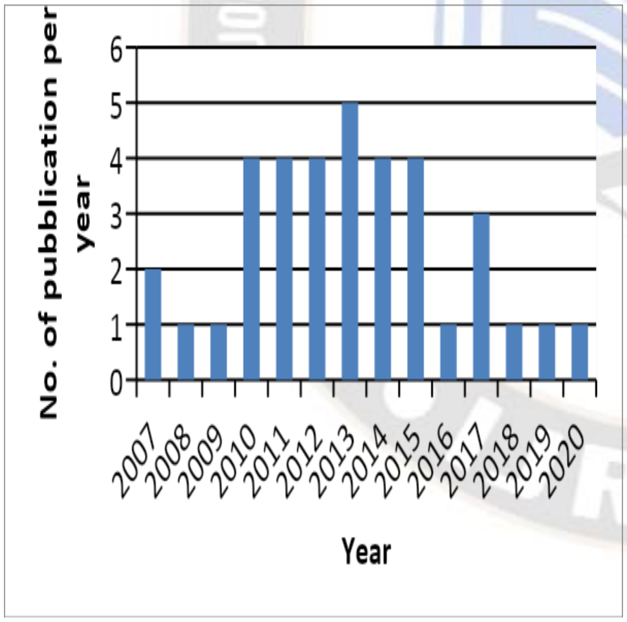

Figure 2: Frequency of publications per year

Input Model-wise distribution of studies

Fig. 3 displays the final model-level results of the literature review. in this figure research parers are categories as input model which shows that most of the researchers used activity diagrams and sequence diagrams for the generation of test cases.

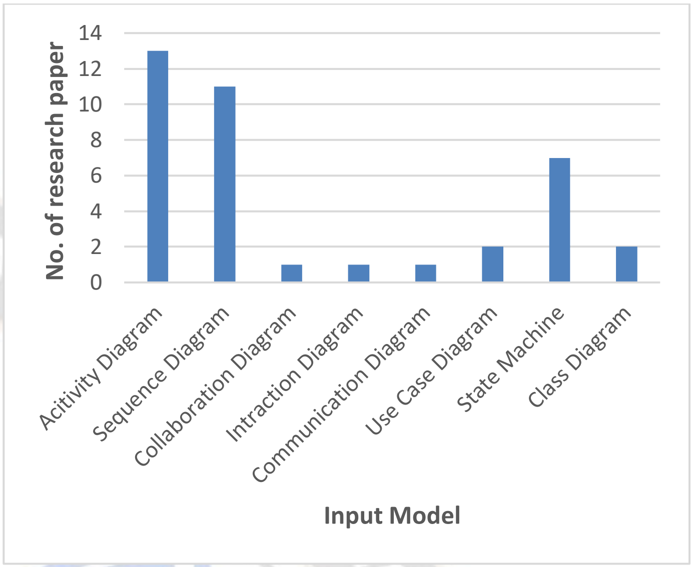

Figure 3: Input model-wise results of the literature review

Research Approach wise distribution of studies

Fig. 4 displays the variation in methodology used by primary research. The study of UML MBT is reflected in the rising popularity of graph-based approaches (42%) as well as heuristic-based techniques (32%).

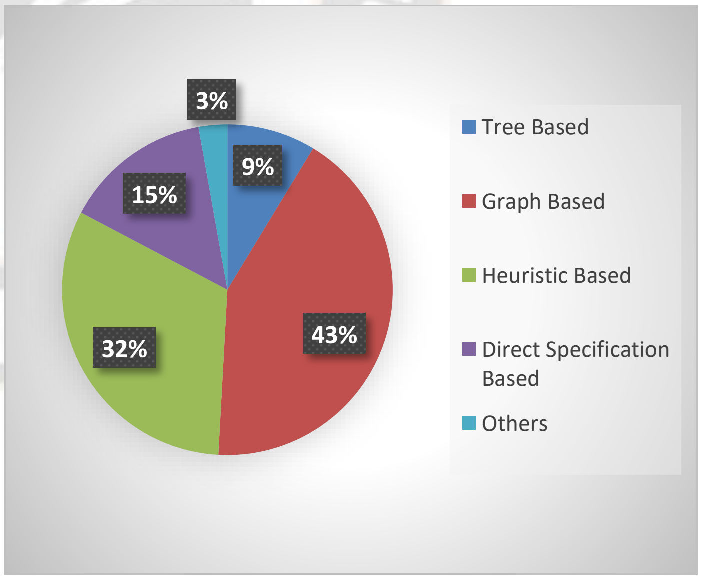

Figure 4: Distribution of primary studies according to the research approach

Systems investigated

To address the study issues stated in the preceding parts, this section presents the findings that were taken from a few primary studies.

RQ1: What are the Different methods Used in UML MBT?

RQ1.1 Tree-Based methods

A linked acyclic graph is a tree. The tree's nodes and connections follow a specific order from the trunk to the leaves. Create test cases from the input model using graph-theoretic tree-based test case generation techniques. The tree-based method first converts the input model into a tree as an intermediate form before using a tree traversal technique to generate test cases from it.

RQ1.2 Graph-Based methods

G = V, E defines a graph as an ordered pair. Each component of this pair, V, represents a collection of nodes or vertices, and E, a number of edges. When e = x on a directed graph, the direction of the edge is from x to y. From a testing perspective, looking for paths in a graph representation of the UML model and testing data values that satisfy constraints along those pathways is essential. Converting flowcharts into their appropriate graph representations is an example of a graph-based method. This includes the translation of activity diagrams into activity diagram graphs and sequence diagrams into sequence diagram graphs. We then utilize a specialized version of depth-first search to generate test cases from each graph (DFS). Afterwards, a system testing graph is derived by combining information from the activity and sequence diagram graphs. The system testing graph already contains the data needed to build the test cases, so testing can be organized around it.

RQ1.3 Heuristic-Based methods

Common meta-heuristic approaches in UML modelling include ant colony optimization and the genetic algorithm. The scale and complexity of the SUT makes thorough testing impractical, hence the employment of genetic algorithms in search strategies is encouraging. An evolutionary algorithm, or genetic algorithm, is modeled after the method of natural selection. The genetic algorithm is often utilized to develop efficient optimization solutions. Over and over, a genetic algorithm improves a pool of applicants and the fitness function until it meets the set of conditions for breadth of coverage [21]. Concurrency [S20] [S21] makes activity diagram-based test case generation difficult. The fitness function creates full, concurrent, and fully feasible pathways by utilising the active node list and action script associated with each transition node.

RQ1.4 Direct UML Specification Processing

In direct UML specification processing method models are not converted into an intermediate form such as graph and tree, researcher directly generating test cases from any tool which they implement.

Table 3: Review summary

Reference	Approach	Input Model	Methodology	Intermediate Forms	Testing Type	Case Study
[S1]	Tree Based	Activity diagram	Condition Classification tree method	Condition classification tree	System testing	Yes
[S2]	Tree Based	Activity diagram	Extenics	Euler Circuit	Euler path	No
[S3]	Tree Based	Communication diagrams	Post-order traversal	communication diagram	Path Testing, Boundary Coverage	No
[S4]	Graph Based	Activity diagram, sequence diagram	DFS algorithm	Activity graph, sequence graph, SYTG	Integration testing	Yes
[S5]	Graph Based	Activity diagram	Business flow control DFS	Activity graph	System testing	No
[S6]	Graph Based	Activity diagram	DFS traversal	Intermediate testable model	System testing	Yes
[S7]	Graph Based	Activity diagram	DFC, algorithm	Activity flow graph	System testing	Yes

[S8]	Graph Based	Interaction diagram	Slicing of diagram	MDG	Integration testing	No
[S9]	Graph Based	Sequence diagram	DFS	Labelled transition diagram	Integration testing	Yes
[S10]	Graph Based	Sequence diagram	TCG algorithm	-	Path coverage	Yes
[S11]	Graph Based	Sequence diagram	DFS traversal	Structured control graph	All paths	No
[S12]	Graph Based	State Machine	Data Flow Analysis	EAFG	Path testing	Yes
[S13]	Graph Based	Collaboration Diagram	Prism and Dijktras Algorithm	CWG	System Testing	Yes
[S14]	Graph Based	Sequence Diagram	BFS DFS	Concurrent Composite graph	Message sequence path testing	No
[S15]	Graph Based	Sequence Diagram		SDT, SDG	System Testing	Yes
[S16]	Graph Based	Sequence Diagram	SLT	SG, SLT Stimulus path SLT stack	System Testing	No
[S17]	Graph Based	Activity Diagram	Transformation Based approach	UAD graph, Extended AND-OR tree	Concurrency	Yes
[S18]	Graph Based	Sequence, State Machine Diagram	Coupling based, Data flow testing	Control flow Graph	Integration Testing	Yes
[S19]	Heuristic	State chart diagram, sequence diagram	Genetic algorithm	SCG, SG, SYTG	System testing	Yes
[S20]	Heuristic	Activity diagram	Evolutionary algorithm genetic algo	ECFG	Path Coverage	No
[S21]	Heuristic	Activity diagram	Genetic algorithm	AFT, AFG	System testing	No
[S22]	Heuristic	Activity diagram	Hybrid genetic algorithm	diagram of control flow	Path testing	Yes
[S23]	Heuristic	State chart diagram	Genetic algorithm	Graph of Control and Its Expansion	-	No
[S24]	Heuristic	Activity diagram	Orientation Based Ant Colony optimisation		Path Testing	No
[S25]	Heuristic	Sequence Diagram, Machine Diagram	XMI parser		Integration Testing	No
[S26]	Heuristic	Sequence Diagram	Genetic Algorithm		Prime Path testing	No
[S27]	Heuristic	State Machine	Test Case Generator	FTS	Conformance	No

[S28]	Heuristic	Sequence Diagram	Real time fault driven stress methodology		Stress	Yes
[S29]	Heuristic	Sequence Diagram	Conformance checking approach	Generate Sates, transitions, Simplify	System Testing	Yes
[S30]	Direct UML specification	Use Case, Activity Diagram	TDD	AOAD, Test sequence	System Testing	No
[S31]	Direct UML specification	Class, Use Case & Activity Diagram	Agent-based regression	XMI format	Regression testing	Yes
[S32]	Direct UML specification	UML Model	Parsing token and matching mechanism	Pattern Class Name		Yes
[S33]	Direct UML specification	State machine	C#	Atomic state	System testing	No
[S34]	Direct UML specification	Class diagram	OCL code	-	System testing	Yes
[S35]	Direct UML specification	State machine	Papyrus-RT Modeling IDE	-	Concolic testing	Yes

RQ2: What are the UML models are used in different methods in the primary study?

UML models may be broken down further into two types: structural and behavioral. While structural models are rigid and usually only address the high-level organization of SUT, most researchers have turned to behavioral models to round out their recommended strategy. 32 out of 35 primary study works on behavioral models. It includes activity, sequence, interaction, and state chart diagrams. Just three out of thirty-five main works rely on the class diagram as their structural paradigm. Due to the need for more specifics on model implementation, behavioral models are often preferred by academia. [20].

RQ3: Is the intermediate form is used in the primary study?

The intermediate form is the stage in which a UML diagram is connected to a graph or any other format used to generate the final test case. Our study discovered 6 of 35 recommended methods functioned without using any intermediate form throughout the test case generation process. Control flow graphs and sequence graphs are only two examples of the medium formats that may be used to construct test cases. This is especially the case for structural models, where the original UML diagrams may not allow for easy extraction of test cases. Since most intermediate formats are trees or graphs, most approaches employ a tree-search-based algorithm, such as breadth-first search or depth-first search, to walk the tree and generate test cases.

RQ4: Which testing level is achieved in the primary study?

Unit testing, integration and system testing, system testing, and acceptance testing are the four pillars of software testing.[1]. 13 out of 35 primary studies perform system-level testing; system testing is to validate end-to-end system specifications that are described in the software requirement specifications. Some researchers perform integration testing, path testing, regression testing and stress testing. 3 out of 35 primary studies do not specify any testing type.

RQ5: What methodology was used in the primary study?

Most of the original research we investigated employed networks as an intermediary form, and those studies created test cases from the models provided using graph traversal algorithms. 9 out of 35 works on graph traversal algorithm. Test case generation and optimization using an ant colony algorithm with an orientation-based strategy has been the focus of a number of primary research studies, as have other approaches like the genetic algorithm, the transition-based approach, the condition classification method, and test driven development.

RQ6: Is the primary study including case study?

We found out that majority of researchers utilised case studies to illustrate the output of their models. Researchers may examine their results in further detail using a case study technique, but only in relation to a certain place or topic. Typically, a case study approach will take a small sample size of data and demonstrate the outcome. In its truest form, case studies examine and provide in-depth contextual analysis of a select few circumstances or occurrences as well as the connections between them. [22].

DISCUSSION

Due to the sheer amount of potential inputs and processes, evaluating every single conceivable test case is impractical for most applications. The most difficult part of testing software is deciding on a collection of test cases to uncover all of the bugs in the SUT. The basic objective of Model-Based Testing (MBT) is to automate the production and running of test cases generated from UML models. More test coverage, quicker mistake detection, and a shorter minimum standard are the main advantages of MBT. The test cases for MBT were developed using a variety of different models. Here, we have zeroed in on UML MBT's test cases and testing procedure, focusing on how they may be generated with the help of UML models.

UML models may be used to represent the system in a straightforward manner with greater abstraction. We conducted an SLR on UML MBT to investigate the existing approaches and the unanswered questions in this study area. Out of the 65 publications we read, we found 35 that directly addressed our study topics. Thus, we consider them to be primary studies. Considering the varied research and the history of model-based testing, we developed an intermediate type of model to establish the level of model-based testing. Several conclusions about model-based testing may be drawn from our research.

We have developed a foundational understanding of UML MBT and its driving forces. Models are used to create test cases, which can then be used to examine working software. Since there is agreement on the general model-based testing procedure, all of the methods detailed in the main paper can be understood as instantiations of the reference process model. The methodologies vary in how UML models are chosen, how intermediate forms are generated, how test criteria are described, how test cases are generated, what kind of testing is performed, and what kind of case study is used. We have provided a table-based presentation of these components.

In addition, we have recognised a number of challenges in developing efficient test cases. The majority of academics have seen graphs as the intermediate representation of UML diagrams and have limited their attention to just path coverage requirements. Most researchers use activity and sequence diagrams as the input models. However, there is no framework that can encompass all forms of UML diagrams, produce test cases, and prioritise the test cases.

CONCLUSION

Testing real-world systems manually or exhaustively is unrealistic due to the sheer amount of possible input combinations and actions. Creating test cases that can identify defects in the SUT is a massive obstacle. MBT testing is initiated early in the product development process to cut down on testing time. UML is now widely accepted as the gold standard for software modelling in both the academic and business worlds. This motivates the need for an SLR on UML-based model-based testing strategies.

The results of our research show that the input model, methodology, and intermediate form of UML MBT. We have looked at a few methods for creating test cases, but there are likely many more that may be derived from these. Research into these methods might help UML MBT get the most possible advantages. Researchers may find this study's findings helpful as a starting point for further research of UML MBT.

REFERENCES

[1] Aditya P. Mathur, 'Foundations of Software Testing' 2nd Edition by Pearson, 2013.
[2] Dr. Leelevathi Rajamanicham, 2014 'Testing Tool for Object Oriented Software', International Journal of Scientific Research and management.
[3] John D. Mcgregor, David A. Sykes 2001 'A practical guide to testing object- oriented software',3rd edition by Addison-Wesley.
[4] Sanjeev Patwa, Anubha Jain 2016 'Effect of Analysis and Design Phase factors on Testing of Object Oriented Software', IEEE.
[5] Namita Khurana, R. S. Chillar 2014 'Literature Review of Test Case generation Techniques for Object Oriented System', International Journal of Computer Application, vol. 105-no. 15.
[6] P.D. Ratna Raju, Suresh Cheekaty, Harishbabu Kalidasu 2011 'Object oriented Software Testing', International Journal of Computer Science and Information Technology, Vol. 2.
[7] Daniel Maciel, Ana C.R. Paiva, and Alberto Rodrigues Da Silva. 2019. 'From requirements to automated acceptance tests of interactive apps: An integrated model-based testing approach', ENASE 2019, pp.265–272.
[8] Marvin V Zelkowitz, 'Perspectives in software engineering. ACM Computing Surveys' CSUR 10, 2 1978, 197–216.
[9] Arilo C. Dias Neto, Rajesh Subramanyan, Marlon Vieira, and Guilherme H. Travassos, 2007 'A survey on model-based testing approaches: A systematic review', ASE. pp. 31–36.

[10] Chunhui Wang, Fabrizio Pastore, Arda Goknil, Lionel Briand, and Zohaib Iqbal, 2015 'Automatic generation of system test cases from use case specifications', ISSTA 2015, ACM Press, New York, New York, USA, Vol. 18. pp.385–396.
[11] Harendra Singh, Mohammad Arif, MohdFayak, Afsaruddin Khan, 2016 'A Survey on Object-Oriented Software Testing', International Journal of Computer Science and Information Technology.
[12] V. Garousi, MikaV. Mäntylä, 2016 'When and what to automate in software testing? A multi-vocal literature review', Inf. Software Technol. 76, pp.92–117.
[13] H. Muccini, A. Bertolino, P. Inverardi, 2003 Using software architecture for code testing, IEEE Trans. Software Eng. 29, pp.160–171.
[14] Md khalid Hussain, Dr.Krisna Prasad, 2016 'A Literature Survey for Test Case Generation Using UML Model', International Journal of Computer Science Trends and Technology IJCST, Vol- 4.
[15] Paramjit Kaur, Rupinder Kaur, 2013 'Approaches for Generating Test Cases Automatically to Test the Software', International Journal of Engineering and Advanced Technology.
[16] Aliya Hussain, Saurabh Tiwari, Jagadish Suryadevara, and Eduard Enoiu, 2018 'From modeling to test case generation in the industrial embedded system domain', Lecture Notes in Computer Science, pp. 499–505.
[17] Hartman, M. Katara, S. Olvovsky, 2006 'Choosing a test modeling language: a survey', doi:10.1007/978-3-540-70889-6, vol.16.
[18] Paramjit Kaur, Rupinder Kaur, 2013 'Approaches for Generating Test Cases Automatically to Test the Software', International Journal of Engineering and Advanced Technology.
[19] Kunxiang Jin, Kevin Lan, 2021 'Generation of Test Cases from UML Diagrams - A Systematic Literature Review', SEC 2021.
[20] Woo Yeol Kim, Hyun Seung Son, and Robert Young Chul Kim, 2011 'A study on test case generation based on state diagram in modeling and simulation environment', CCIS.
[21] Mahesh Shirole, Rajeev Kumar, 2013 'UML Behavioural Model Based Test Case Generation: A Survey', ACM SIGSOFT, vol. 38, no.4.
[22] Zaidah Zainal 2007 'Case study as a research method', Journal Kemanusiaan bil.9.
[23] Tanwir Ahmad, Junaid Iqbal, Adnan Ashraf, Dragos Truscan, Ivan Porres, 2019' Model-based testing using UML activity diagrams: A systematic mapping study', Computer Science Review, Vol. 33.
[S1] Supaporn Kansomkeat, Jeff Offutt, 2010 'Generating Test Cases from UML Activity Diagram using the Condition-Classification Tree Method', IEEE, 2nd International Conference on Software Technology and Engineering, Vol.1, pp.62-66.
[S2] Liping Li, Xingsen Li, Tao He, JieXiong, 2013' Extenics-based Test Case Generation for UML Activity Diagram', ELSEVIER, Procedia Computer Science, oi: 10.1016/j.procs.2013.05.151, pp.1186-1193.
[S3] Philip Samuel, Rajib Mall, Pratyush Kanth 2007 'Automatic test case generation from UML communication diagrams', Information and Software Technology 49, doi:10.1016/j.infsof.2006.04.001, Pp.158–171.
[S4] Meiliana, Irwandhi, Ricky, Daniel, 2017 'Automated Test Case Generation from UML Activity Diagram and Sequence Diagram using Depth First Search Algorithm', ELSEVIER, ICCSCI doi: 0.1016/j.procs.2017.10.029, pp.629-637.
[S5] Walaithip, Suwatchai, Luepol, 2016 'Generating Test Cases from UML Activity Diagram Based on Business Flow Control', ACM, ICNCC, DOI: http://dx.doi.org/10.1145/3033288.3033311, pp.155-160.
[S6] Nayak and D. samanta, 2011 'Synthesis of Test scenarios using UML Activity Diagram', Software and System Modelling, vol. 10, DOI 10.1007/s10270-009-0133-4.
[S7] Ranjita K. Swain, Vikas Panthi, Prafulla K. Behera, 2013 'Generation of Test Cases using Activity Diagram', International Journal of Computer Science and Informatics, vol. 3, Issue-2.
[S8] Ranjita K. Swain, Vikas Panthi, Prafulla K. Behera, 2012 'Test Case Design Using Slicing of UML Interaction Diagram', ELSEVIER, Procedia Technology.
[S9] Emanuela G. Cartaxo, Francisco G. O. Neto, D. L. Machado, 2007 'Test Case Generation by means of UML Sequence Diagrams and Labelled Transition System', IEEE, CNPq/Brazil Process 550466/2005-3.
[S10] Nabilah K. Bahrin, Radziah Mohamad, 2015 'TCG Algorithm Approach for UML Sequence Diagram', Malaysian Software Engineering Conference, 978-1-4673-8227-4/15/$31.00, pp.43-48.
[S11] Nayak and D. Samanta, 2010 'Automation Test Data Synthesis using UML Sequence diagrams', Journal of Object Technology, Vol. 9, no. 2, pp. 75–104.
[S12] Lionel Briand1, Y. Labiche2 and Q. Lin2 2010 'Improving the coverage criteria of UML state machines using data flow analysis', Software Testing, Verification and Reliability Software, .
[S13] Syed Usman Ahmed, Sneha Anil Sahare, Alfia Ahmed, 2012 'generate test cases from the input model', World Journal of Science and Technology, vol.1, pp.4-6.
[S14] Monalisha Khandai, Arup Abhinna Acharya, Durga Prasad Mohapatra, 2011 'A Novel Approach of Test Case Generation for Concurrent Systems Using UML

Sequence Diagram', 978-1-4244-8679-3/11/2011 IEEE, pp.157-161.

[S15] S. Shanmuga Priya, P. D. Sheba Kezia Malarchelvi, 2013 'Test Path Generation Using UML Sequence Diagram', International Journal of Advanced Research in Computer Science and Software Engineering, Vol. 3, Issue 4, pp.1069-1076.

[S16] Mani P., Prasanna M., 2017' Test Case Generation for Embedded System Software Using Uml Interaction Diagram', Journal of Engineering Science and Technology, Vol. 12, No. 4, pp.860-874.

[S17] Chang-ai Sun, Yan Zhao, Lin Pan, Xiao He, and Dave Towey, 2014 'A Transformation-based Approach to Testing Concurrent Programs using UML Activity Diagrams', Software—Practice and Experience Wiley InterScience DOI: 10.1002/spe.

[S18] Lionel Briand, Yvan Labiche, and Yanhua Liu, 2012 'Combining UML Sequence and State Machine Diagrams for Data-Flow Based Integration Testing', A. Vallecillo et al. Eds.: ECMFA 2012, LNCS 7349, pp. 74–89.

[S19] Namita Khurana, R. S. Chillar, 2015 'Teat case Generation and Optimization using UML Models and genetic Algorithm', 3rd International Conference on Recent Trends in Computing, doi: 10.1016/j.procs.2015.07.502, pp.996-1004.

[S20] Mahesh Shirole, Mounika Kommuri, Rajeev Kumar, 2012 'Transition Sequence Exploration of UML Activity Diagram using Evolutionary Algorithm', Proceedings of ISEC, ACM 978-1-4503-1142-7/12/02, pp.97-100.

[S21] Ajay k Jena, Santosh K Swain, Durga P. Mohapatra, 2014 'A Novel Approach for Test Case Generation from UML activity Diagram' IEEE, ICICT, 978-1-4799-2900-9/14, pp.621-629.

[S22] Xinying Wang, Xiajun Jiang, Huibin Shi, 2015 'Prioritization of Test Scenarios using Hybrid Genetic Algorithm Based on UML Activity Diagram', IEEE Software, pp.854-857.

[S23] Mahesh Shirole, Amit Suthar, Rajeev Kumar, 2011 'Generation of Improved Test Cases from UML State Diagram Using Genetic Algorithm' ACM ISEC, ACM 978-1-4503-0559-4/11/02, pp.125-134.

[S24] Vinay Arora, Maninder Singha, Rajesh Bhatia, 2020 'Orientation-based Ant colony algorithm for synthesizing the test scenarios in UML activity diagram', Information and Software Technology, https://doi.org/10.1016/j.infsof.2020.106292.

[S25] Dominykas Barisas, Eduardas Bareiša, and Šarūnas Packevičius, 2013 'Automated Method for Software Integration Testing Based on UML Behavioral Models', ICIST 2013, CCIS 403, pp. 272–284.

[S26] Bahare Hoseini, Saeed Jalili, 2014 'Automatic Test Path Generation from Sequence Diagram Using Genetic Algorithm', 7th International Symposium on Telecommunications, 978-1-4799-5359-2/14/$31.00 ©2014 IEEE, pp.106-111.

[S27] Dirk Seifert, 2008 'Conformance Testing based on UML State Machines: Automated Test Case Generation, Execution and Evaluation', Dirk Seifert. Conformance Testing based on UML State Machines: Automated Test Case Generation, Execution and Evaluation. Research Report.

[S28] Vahid Garousi, 2009 'Fault-driven stress testing of distributed real-time software based on UML models', Software Testing, Verification and Reliability Software, Wiley Online Library, DOI: 10.1002/stvr.418, pp.101-124.

[S29] João Pascoal Faria, Ana C.R. Paiva, and Mário Ventura de Castro, 2013 'Techniques and Toolset for Conformance Testing against UML Sequence Diagrams', IFIP International Federation for Information Processing, ICTSS 2013, LNCS 8254, pp. 180–195.

[S30] Saurabh Tiwari, Atul Gupta, 2015 'An Approach of Generating Test Requirements for Agile Software Development', ISEC'15, http://dx.doi.org/10.1145/2723742.2723761, pp.186-195.

[S31] Pradeep Kumar Arora, Rajesh Bhatia, 2017 'Agent-Based Regression Test Case Generation Using Class Diagram, Use Cases and Activity Diagram', ICSCC, Procedia Computer Science 125, pp.747–753.

[S32] Ashish Verma, Dr Maitrayee Dutta, 2014 'Automated Test case generation using UML diagrams based on behaviour', international Journal of innovation in Engineering and Technology, Vol. 4 Issue 1, pp. 31-39.

[S33] Ricardo D. F. Ferreira, P. Faria, C. R. Paiva, 2010 'Test Coverage Analysis of UML State Machines', IEEE 3rd International Conference on Software Testing, Verification, and Validation, DOI 10.1109/ICSTW.2010.60, pp.284-289.

[S34] Kalou Cabrera Castillos, Fr'ed'eric Dadeau, Jacques Julliand, and Safouan Taha, 2011 'Measuring Test Properties Coverage for Evaluating UML/OCL Model-Based Tests', IFIP International Federation for Information Processing, pp. 32–47.

[S35] Reza Ahmadi, Karim Jahed, Juergen Dingel, 2019' mCUTE: A Model-level Concolic Unit Testing Engine for UML State Machines', 34th IEEE/ACM International Conference on Automated Software Engineering ASE, DOI 10.1109/ASE.2019.00132, pp.1182-1185.