The Use of UML Use Case Diagrams and Their Specifications for Test Case Generation

Ieva Valatkevičiūtė [1], Lina Čeponienė [1] and Tomas Valatkevičius [1]

[1] *Faculty of Informatics, Kaunas University of Technology, Studentų St. 50, Kaunas, 51368, Lithuania.*

Abstract

Manual test cases remain an important part of software testing, but their creation is often time-consuming and requires significant effort. In this paper, we propose a method for generating manual test cases based on UML use case diagrams and their specifications. The approach uses structured information from use case specification tables, which are created based on an adapted Volere template, and allows the user to participate in the generation process by selecting relevant test scenarios. A test case generation tool implementing the proposed method was developed using Python. The tool performs an initial analysis of the uploaded specification, identifies use cases, and generates test cases according to predefined rules. Evaluation results demonstrate that the proposed approach can successfully generate structured test cases from UML use case specifications and thus support the software testing process.

Keywords

UML, use case specification, manual testing, test cases.

1. Introduction

Software development is a complex process that includes analysis, design, implementation, and testing stages. Each of these stages plays an important role in ensuring software quality. One of the major challenges is achieving effective and reliable testing that ensures a system functions according to its specifications and meets user expectations. Despite the increasing adoption of automated testing techniques, manually designed test scenarios remain widely used in software testing practices. Manual test cases allow testers to evaluate complex user interactions, explore unexpected system behaviors, and validate usability aspects that may be difficult to capture through automated approaches. In many real-world projects, manual testing complements automated methods and remains an important part of quality assurance processes. Studies indicate that human-designed test scenarios continue to play a significant role in detecting defects and ensuring system reliability, particularly in cases requiring exploratory testing or domain knowledge [1]-[3]. In projects where test case specifications are required, test cases are still created manually. This approach requires significant time and human resources, and due to the large number of possible user interactions and unexpected system responses, sufficient test coverage is not always achieved. As a result, manually designed test cases may overlook rare or complex scenarios, leaving parts of the system insufficiently tested and defects undetected until later stages of development [4].

As software systems become larger and more complex, traditional manual testing approaches become increasingly inefficient. Therefore, many organizations seek ways to automate the generation of test cases. Automated testing solutions allow faster and more systematic generation of testing scenarios, improve coverage, and reduce the need for extensive human effort [4]. One possible approach is the use of UML models, particularly use case diagrams and their specifications, which describe system functionality and user interactions. These models help identify system requirements and can serve not only as design artifacts but also as a valuable source for deriving test cases [5].

CEUR-WS.org/Vol-4317/paper8.pdf

*IVUS 2026: Information Society and University Studies 2026, May 12, Kaunas, Lithuania

ieva.valatkeviciute@ktu.edu (I. Valatkeviciute); lina.ceponiene@ktu.lt (L. Ceponiene); tomas.valatkevicius@ktu.lt (T. Valatkevicius)

0000-0002-6812-0215 (L. Ceponiene)

Copyright © 2026 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0)

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

published 2026-11-06

However, existing tools for generating test cases from UML models often provide limited capabilities, lack accuracy, or require complex configurations and specialized knowledge [6]. This creates a need for more flexible and reliable approaches for automated test case generation based on UML use case diagrams and their specifications. The aim of this research is to facilitate the software testing process by proposing and implementing a method that automatically generates test cases from UML use case diagrams and their specifications. The proposed methodology was implemented as a prototype tool and experimentally evaluated, demonstrating its ability to successfully generate structured test cases.

The rest of the paper is structured as follows. Section 2 provides background and reviews related work, including UML use case specifications, test case design principles, and existing tools for test case generation. Section 3 presents the proposed methodology for generating test cases from UML use case diagrams and their specifications. Section 4 describes the implementation of the developed tool and the results of its evaluation. Finally, Conclusions section summarizes the findings and outlines directions for future work.

2. Background

The section examines the role of UML use case models in the test design process, the principles and methods of test case construction, and the capabilities of existing tools that generate test cases from UML artifacts. Particular attention is given to the limitations of current solutions, which often lack accuracy, require complex configurations, or provide insufficient flexibility for real-world projects. These shortcomings highlight the need for a more reliable and structured approach to generating test cases from both UML use case diagrams and their detailed specifications.

2.1. Use Case Specifications as an Input for Test Case Generation

Unified Modeling Language (UML), standardized by the Object Management Group (OMG), is widely used for visual modeling and design of software systems [7]. Among various UML diagram types, use case diagram is one of the most commonly applied for representing functional system requirements. It illustrates interactions between actors and the system, helping to identify the actions the system must perform to satisfy user needs. A typical use case diagram consists of actors, use cases, and relationships that describe how external entities interact with system functionality. Additional relationships such as include and extend, as well as association multiplicities, provide a more detailed description of system behavior and dependencies between components [8]. Due to its clear and standardized structure, the use case diagram is widely used to communicate system functionality and ensure that all stakeholders share a common understanding of system requirements [8].

While use case diagrams provide a high-level view of system functionality, detailed behavior is described through use case specifications. A use case specification defines how the system should behave when a particular use case is executed and typically includes elements such as the goal, actors, preconditions, main scenario steps, and alternative scenarios [9]. In many projects, these specifications are structured using templates such as the Volere framework [10], which provides a consistent format for documenting requirements and ensuring their completeness [11]. A well-structured use case specification reduces ambiguity, facilitates communication among analysts, developers, and testers, and supports systematic derivation of test scenarios. Therefore, use case specifications play an important role in software development processes and can serve as a reliable source for generating test cases based on system requirements [12].

2.2. Manual Test Cases and Approaches for Their Development

Test cases are structured scenarios used to verify how a system responds to specific inputs or conditions [13]. Test cases are an essential component of the quality assurance (QA) process, as they support early defect detection and help reduce risks during software deployment and maintenance [14]. In addition to functional validation, test cases may also address reliability, performance, and security aspects of a system. A well-defined test case structure usually includes input data, execution

steps, expected outcomes, and testing environment information, and may also contain additional attributes such as priority, related requirements, and execution status to facilitate systematic evaluation of software quality [15].

Various methods are applied for designing test cases. Equivalence Partitioning [16] divides input data into groups with similar behavior in order to reduce the number of required tests while maintaining adequate coverage. Boundary Value Analysis [17] focuses on identifying defects at the limits of input ranges, where errors frequently occur. Decision Table Testing [18] is used to analyze complex combinations of conditions and actions, ensuring systematic evaluation of possible scenarios. Another widely used approach is Use Case Analysis [12], where UML use case diagrams and their specifications serve as a structured basis for deriving functional test cases directly from system requirements. Recently, artificial intelligence techniques, including large language models (LLMs), have also been explored for automated test case generation by analyzing textual requirements and identifying user actions and system responses [19]. Although such approaches can reduce manual effort and increase testing coverage, their effectiveness strongly depends on the quality of input data and proper model configuration [20].

With the rapid development of information technologies, numerous methods and tools have been proposed to automate software testing processes using requirements specifications, system models, or use cases. Such solutions aim to reduce manual effort, increase testing efficiency, and improve software quality. Several model-based testing tools have been developed to automatically generate test cases from system models. Examples include Spec Explorer [21] and RT-Tester [22], [23], which derive automated tests from behavioral system models and support model-based testing approaches.

One widely used model-based testing solution is ConformIQ [24], which supports automatic test generation from system models, including UML use case and activity diagrams. The tool analyzes system behavior, identifies possible execution paths, and automatically generates corresponding test scenarios [24].

Another tool used for automated test generation is TestOptimal [25], which applies model-based testing techniques to derive test cases from system behavior models and requirements specifications. The tool allows testers to visualize system behavior using state-based models and automatically generate test scenarios from them.

In contrast to fully model-based tools, some platforms focus more on manual test case management and organization. For example, TestCaseLab [26] is a test management tool that allows testers to create, manage, and execute manual test cases within a centralized environment. The tool provides functionality for organizing test scenarios, tracking test execution results, and improving collaboration between testing team members.

Although these tools demonstrate the potential of automated test generation and test management, many of them rely on complex modeling techniques or require highly detailed models. As a result, UML use case diagrams and their specifications are often treated only as supplementary information sources. Our approach focuses on generating test cases directly from use case diagram and use case specifications, as these artifacts are already present in the development process and require minimal additional effort for preparing them as an input for test case generation.

3. Methodology for Deriving Test Cases from Use Case Specifications

Based on the analysis of existing approaches for automated test case generation, a methodology was developed to support the creation of manual test cases using UML use case diagrams and their specifications. The proposed approach uses structured information from use case diagram and allows the user to participate in the process by reviewing and selecting relevant scenarios before test generation. This methodology aims to simplify the test design process while ensuring that generated test cases reflect both the functional requirements and possible system behaviors. The overall workflow of the proposed approach is presented in **Figure** 1.

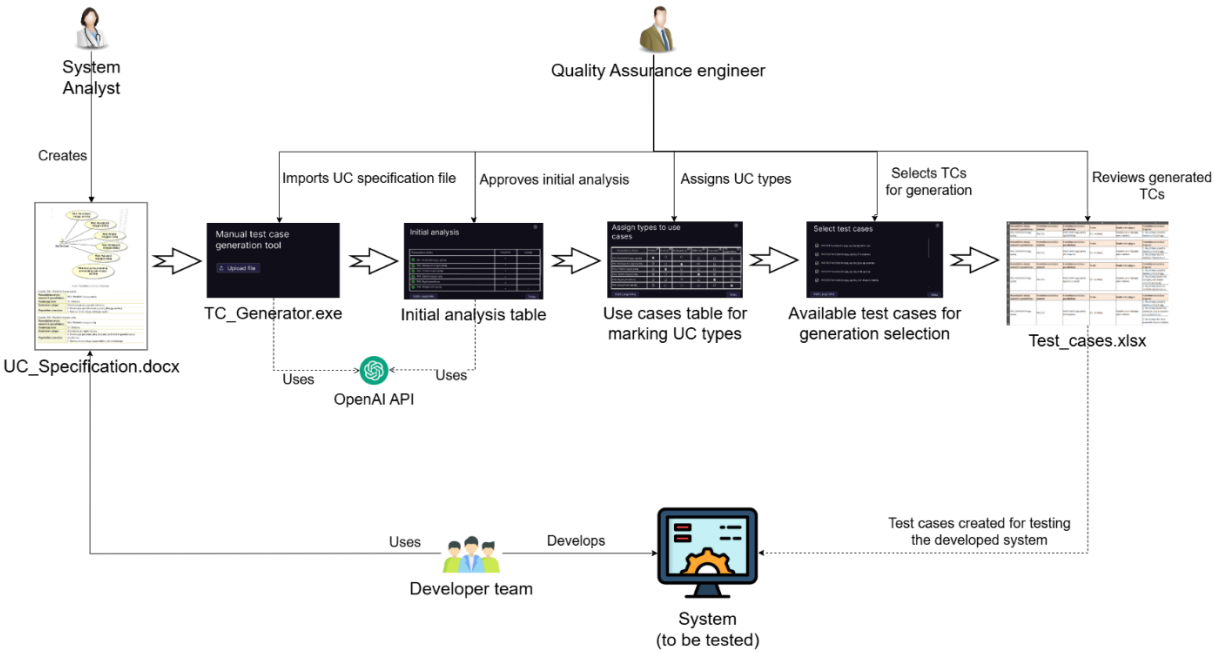


Figure 1: The general workflow diagram of manual test cases generation methodology

The diagram illustrates the overall workflow of the proposed test case generation methodology and the interaction between different stakeholders and system components. The process begins with the preparation of the use case specification file, which contains UML use case diagrams and their detailed specifications. This document is created by a system analyst using an adapted Volere-based template for use case specifications and serves as the main input for the test case generation tool. The specification file is then uploaded to the test case generation tool, where an initial analysis is performed to verify whether the document contains the required structure and information for further processing.

After the initial analysis stage, the quality assurance engineer interacts with the tool by reviewing the identified use cases and assigning specific types (e.g., viewing, creation, editing, deletion, system, or main only). These types determine the set of additional predefined test scenarios that will be included in the generation process. For example, the viewing type adds scenarios such as sorting, pagination, and invalid permissions, while creation, editing, and deletion types add invalid permission scenarios. System and main-only types do not introduce additional scenarios. Based on these selections and the information extracted from the specification, the system generates a set of manual test cases. The final result of the process is a structured collection of generated test cases that can be exported and used in the software testing process.

To define the structure of the input data used by the proposed method, a class diagram was created to represent the elements contained in the use case specification file and the relationships between them. The diagram illustrates how the information from the use case specification document is organized and used during the test case generation process. The diagram is illustrated in **Figure** 2.

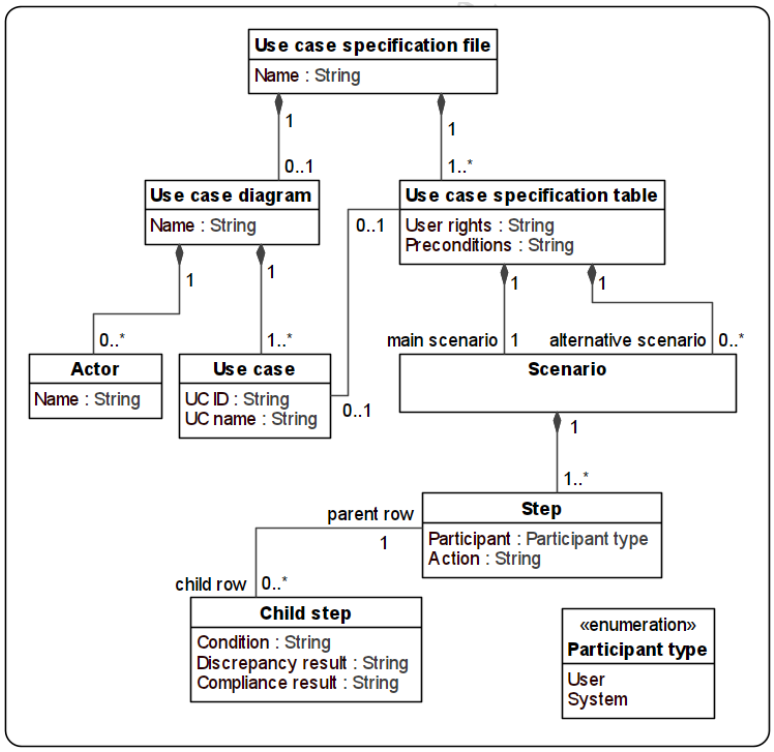


Figure 2: Use case specification file structure

As shown in the diagram, the use case specification file, structured according to an adapted Volere template, may contain a use case diagram and one or more use case specification tables. While the diagram is optional, the specification tables are mandatory because they contain the information required for test case generation. Each specification table describes a single use case and includes elements such as user rights, preconditions, and scenarios. A use case must contain one main scenario, while alternative scenarios may or may not be present. Each scenario is composed of multiple steps, which describe actions performed either by the user or the system. In cases where validation is required, a step may contain child steps describing conditions and corresponding outcomes.

To ensure that the uploaded use case specification file can be correctly processed by the proposed method, a set of rules was defined for describing use case specification tables. These rules establish the required structure and formatting of the information contained in the specification, allowing the system to consistently identify actors, actions, conditions, and scenarios used for test case generation. The defined rules for creating use case specification tables are presented in Table 1.

Table 1.
Rules for creating use case specifications

Rule number	Description
UC.R1	Each use case in the uploaded specification file must be described in a separate table.
UC.R2	Each use case table must contain at least the following information: use case number and name, user permissions, preconditions, and the main scenario.
UC.R3	A user action or step must begin with the word “User.”
UC.R4	A system action or step must begin with the word “System.”
UC.R5	If a step includes a validation process, the parent row must contain the phrase “System checks.”
UC.R6	The child rows of a validation step must specify the condition being checked.
UC.R7	If the condition fails, the action must begin with “If not valid.”
UC.R8	If the condition succeeds, the action must begin with “If valid.”
UC.R9	A cancellation scenario must be described as an alternative scenario.
UC.R10	The use case diagram included in the specification file must have the title “Use Case Diagram.”

To define the structure of the generated test case specification file, a class diagram was created. The diagram illustrates the main elements of the test case structure and the relationships between them, showing how test cases and their components are organized within the generated file. The diagram is illustrated in **Figure** 3.

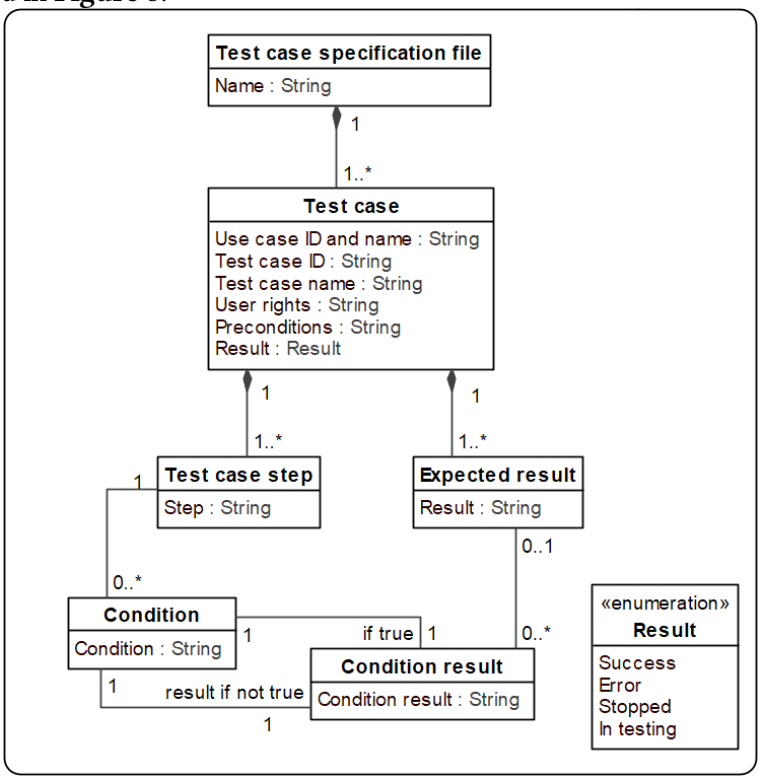


Figure 3**:** Test case specification file structure

As shown in the diagram, the test case specification file contains one or more test cases. Each test case includes information such as the related use case ID and name, test case ID, test case name, user rights, preconditions, and the test result. A test case consists of one or more test case steps and expected results that describe the actions performed and the outcomes that should occur during test execution.

The diagram also includes conditions that may be evaluated during the test. Depending on whether a condition is satisfied, a corresponding condition result is produced. The overall result of a test case can be classified as success, error, stopped, or in testing.

To ensure consistency and clarity in the generated test cases, a set of rules was defined for creating test case specifications. These rules determine how test case identifiers and titles are formed and how the generated test cases are structured. The defined rules for creating test case specifications are presented in Table 2.

Table 2.

Rules for creating test case specifications

Rule number	Description
TC.R1	The test case identifier must follow the template {Use case number}TS{Test case sequence number}.
TC.R2	The sequence number must increment for each generated test case related to the same use case.
TC.R3	The test case title must follow the template {Use case name} ({Tested functionality name}).

The proposed methodology provides a structured approach for generating manual test cases based on UML use case diagrams and their specifications using an adapted Volere template. It ensures consistency by defining rules for input use case specifications and generated test cases, while also allowing user involvement in selecting relevant scenarios.

4. Test Case Generation Tool Implementation

The test case generation tool was implemented as a local desktop application using the Python programming language and supporting libraries for document processing and user interface development. The tool accepts input files in DOCX format containing UML use case specifications structured according to an adapted Volere template. The generated test cases are exported in XLSX format, allowing easy integration with existing testing workflows.

To extract and analyze information from the uploaded file, the tool processes both textual content and embedded diagram images. For identifying use cases from diagrams, the OpenAI API is used as a Large Language Model (LLM) service. The LLM analyzes the diagram image and extracts relevant use case elements, which are then combined with the structured data from specification tables.

Although the application operates locally, it requires communication with the external OpenAI API for diagram analysis. The remaining processing, including validation of the specification structure, scenario extraction, and test case generation based on predefined rules, is performed within the application. This approach enables more advanced analysis capabilities while maintaining the core processing locally. An example of the tool interface is shown in **Figure** 4. The interface is presented in Lithuanian, as it was developed for use in a Lithuanian organizational context.



Figure 4: The example of test generation tool interface

Testing of the developed tool was performed using prepared test data representing small-scale system scenarios. The test cases were based on use case specifications describing typical application functionalities, such as book management (e.g., viewing, creating, updating, and deleting records) and similar CRUD-based operations. The specifications were created according to the defined requirements and structured using the adapted Volere template. Different scenarios were included, such as cases with and without alternative flows, as well as varying use case types, in order to evaluate the tool's ability to handle different input structures. The "Selected use case types" column indicates the categories assigned by the user during the analysis stage, which determine the set of additional test scenarios generated for each use case. The statistics on the input use case specification files used during testing are presented in Table 3.

Table 3.
The input use case specification file's information

Specification file	UC diagram included	UC count in diagram	UC specification tables count	Main scenarios	Alternative scenarios	Selected use case types
F1	Yes	6	6	6	0	Only "Main" scenarios
F2	Yes	6	6	6	1	"Read", "Create", "Update", "Delete", "System".
F3	Yes	5	5	5	0	"Read", "Update".
F4	No	0	6	6	2	"Read", "Create", "Update", "Delete".

Based on the provided input data, the tool generated the corresponding test case files. The summary of the generated test cases is presented in Table 4.

Table 4.
The generated test cases file's information

Specification file	Number of generated workbook sheets	Number of generated TCs						
		Read	Create	Update	Delete	System	Only main	Total
F1	6	0	0	0	0	0	6	6
F2	6	10	3	2	2	1	0	18
F3	5	15	0	4	0	0	0	19
F3	6	10	6	2	2	0	0	20

The results presented in Table 4 demonstrate that the tool successfully generated test cases for all provided input files, adapting to different use case structures and assigned types. The number of generated test cases varies depending on the selected use case types, as certain types introduce additional scenarios. In cases where only main scenarios were selected, the number of generated test cases remained minimal, while more complex configurations resulted in a higher number of generated test cases.

The implementation results demonstrate that the developed tool can effectively process structured use case specifications and generate corresponding test cases in a consistent format. The combination of local processing and OpenAI-based diagram analysis enables handling both textual and visual inputs, extending the applicability of the approach.

5. Conclusions

The results of this research show that UML use case diagrams and their specifications based on Volere format can serve as a source for generating manual test cases. The analysis demonstrated that elements such as preconditions, user actions, and permissions provide sufficient information to derive structured test scenarios. Based on these findings, a methodology was proposed that uses UML use case diagrams and their specifications as the primary input for generating manual test cases. The implemented prototype tool allows users to upload a use case specification file, analyze its structure, and generate test cases according to predefined rules.

The evaluation results showed that the proposed approach is capable of generating structured test cases for various input conditions, including incomplete specifications and cases where use case diagrams are not provided. Nevertheless, the effectiveness of the generated test cases is highly dependent on the quality and completeness of the input data. The proposed solution could be further improved in future work by extending its functionality. Possible improvements include supporting multiple languages in use case specifications, expanding the types of generated test cases, and improving automation of the analysis process.

6. Declaration on Generative AI

The authors have not employed any Generative AI tools.

7. References

[1] M. Viggiato *et al*, "Using Natural Language Processing Techniques to Improve Manual Test Case Descriptions," 2022. Available: https://dl.acm.org/doi/epdf/10.1145/3510457.3513045.

[2] K. Thant and H. H. K. Tin, "THE IMPACT OF MANUAL AND AUTOMATIC TESTING ON SOFTWARE TESTING EFFICIENCY AND EFFECTIVENESS," vol. 3, pp. 88–93, 2023. .

[3] M. D. L. C. Peixoto *et al*, "On the Effectiveness of LLMs for Manual Test Verifications," *arXiv,* 2024. Available: https://arxiv.org/abs/2409.12405. DOI: 10.48550/arXiv.2409.12405.

[4] Qiuying Li, Limeng Zhang and Shuo Liu, "Improving the Efficiency of Software Reliability Demonstration Testing by Introducing Testing Effectiveness," *Symmetry,* vol. 16, *(10),* 2024. Available: https://doi.org/10.3390/sym16101334. DOI: 1334.

[5] G. Booch, J. Rumbaugh and I. Jacobson, "Unified Modeling Language User Guide, The (2nd Edition) (Addison-Wesley Object Technology Series)," *J. Database Manag.,* vol. 10, 1999. .

[6] S. A. Ali Shah *et al*, "Automated Test Case Generation Using UML Class & Sequence Diagram," *Curr. J. Appl. Sci. Technol.,* vol. 15, *(3),* pp. 1–12, 2016. Available: https://journalcjast.com/index.php/CJAST/article/view/1345. DOI: 10.9734/BJAST/2016/24860.

[7] M. Fowler and K. Scott, *UML Distilled: A Brief Guide to the Standard Object Modeling Language.* Addison-Wesley, 2003.

[8] N. Pahwa and K. Solanki, "UML based Test Case Generation Methods: A Review," 2014. Available: https://www.researchgate.net/publication/269669135_UML_based_Test_Case_Generation_Methods_A_Review. DOI: 10.5120/16707-6859.

[9] OMG, "Unified modeling language (UML) specification," Object Management Group, -12. 2017.

[10] Robertson , James and Robertson Suzanne, "Volere Requirements Specification Template," 2012. Available: https://www.cs.uic.edu/~i440/VolereMaterials/templateArchive16/c%20Volere%20template16.pdf.

[11] Cem Kaner, J.D. and Ph.D, "What IS a Good Test Case?" 2003. Available: https://www.techwell.com/sites/default/files/articles/XDD6505filelistfilename1_0.pdf.

[12] H. Wu, "An effective equivalence partitioning method to design the test case of the WEB application," in *2012 International Conference on Systems and Informatics (ICSAI2012),* 2012, . DOI: 10.1109/ICSAI.2012.6223567.

[13] X. Guo, H. Okamura and T. Dohi, "Optimal test case generation for boundary value analysis," *Software Quality Journal,* vol. 32, pp. 1–24, 2024. . DOI: 10.1007/s11219-023-09659-9.

[14] I. Díaz *et al*, "A specification pattern for use cases," *Information & Management,* vol. 41, *(8),* pp. 961–975, 2004. Available: https://www.sciencedirect.com/science/article/pii/S0378720603001502. DOI: 10.1016/j.im.2003.10.003.

[15] B. Dobing, "Use Cases in the UML," .

[16] M. Sholeh *et al*, "Black Box Testing on ukmbantul.com Page with Boundary Value Analysis and Equivalence Partitioning Methods," *Journal of Physics: Conference Series,* vol. 1823, *(1),* pp. 012029, 2021. Available: https://dx.doi.org/10.1088/1742-6596/1823/1/012029. DOI: 10.1088/1742-6596/1823/1/012029.

[17] I. Prayudha, R. Hartani and Y. Divayana, "Boundary Value Analysis Testing Techniques on Learning Management System Applications," *International Journal of Engineering and Emerging Technology,* vol. 4, pp. 31, 2020. . DOI: 10.24843/IJEET.2019.v04.i02.p07.

[18] E. Enoiu *et al*, "Understanding problem solving in software testing: An exploration of tester routines and behavior," in Anonymous 2023, . DOI: 10.1007/978-3-031-43240-8_10.

[19] N. Castillo, A. Güemez and R. Aguilar Vera, "Team Composition in Software Engineering Development Phases: A systematic literature review," *International Journal of Combinatorial Optimization Problems and Informatics,* vol. 15, pp. 237–253, 2024. . DOI: 10.61467/2007.1558.2024.v15i5.584.

[20] W. E. Perry, *Effective Methods for Software Testing, CafeScribe: Includes Complete Guidelines, Checklists, and Templates.* 2007.

[21] Microsoft. (-05-15). *Spec Explorer.* Available: https://learn.microsoft.com/en-us/previous-versions/visualstudio/spec-explorer/ee620411(v=specexplorer.10).

[22] A. Cavalcanti *et al*, "Bringing RoboStar and RT-tester together," in *Applicable Formal Methods for Safe Industrial Products: Essays Dedicated to Jan Peleska on the Occasion of His 65th Birthday*Anonymous 2023, .

[23] M. Baqar and R. Khanda, "The Future of Software Testing: AI-Powered Test Case Generation and Validation," *arXiv Preprint arXiv:2409.05808,* 2024. .

[24] A. Miyazawa[1], U. Schulze and J. Timmis, "Check for updates Bringing RoboStar and RT-Tester," *Applicable Formal Methods for Safe Industrial Products: Essays Dedicated to Jan Peleska on the Occasion of His 65th Birthday,* vol. 14165, pp. 16, 2023. .

[25] Anonymous . *TestOptimal v7.* Available: https://testoptimal.com/v7/wiki/doku.php?id=start.

[26] Z. Alyabroodi *et al*, "A comparison study for test case management tools," in *Artificial Intelligence and Economic Sustainability in the Era of Industrial Revolution 5.0*, A. M. A. Musleh Al-Sartawi and A. I. Nour, Eds. 2024, Available: https://doi.org/10.1007/978-3-031-56586-1_41. DOI: 10.1007/978-3-031-56586-1_41.